НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

«До захисту допущено»

в. о. завідувача кафедри

_____ О.Л. Тимощук

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
з напряму підготовки 6.040303 «Системний аналіз»
на тему: «Рекомендаційна система товарів»

Виконав:
студент IV курсу, групи КА-51
Тарнавський Максим Вікторович _____

Керівник:
доцент, к. ф.-м. н. Дідковська М.В. _____

Консультант з економічного розділу:
доцент, к. е. н. Шевчук О.А. _____

Консультант з нормоконтролю:
доцент, к. т. н. Коваленко А. Є. _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – 6.040303 «Системний аналіз» («Системний аналіз і управління»)

ЗАТВЕРДЖУЮ
в. о. завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Тарнавському Максиму Вікторовичу

1. Тема роботи «Рекомендаційна система товарів», керівник роботи доцент кандидат ф.-м. н., Дідковська Марина Віталіївна, затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи _____
_____
_____
_____
_____
_____

4. Зміст роботи_____
_____
_____
_____
_____
_____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)_____
_____
_____
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	О.А. Шевчук, доцент		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка

Студент _____ М. В. Тарнавський

Керівник роботи. _____ М. В. Дідковська

РЕФЕРАТ

Дипломна робота: 81 с., 7 табл., 27 рис., 2 дод., 21 джерел

РЕКОМЕНДАЦІЙНІ СИСТЕМИ, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, КОНТЕНТНА ФІЛЬТРАЦІЯ, ГІБРИДНА МОДЕЛЬ, ПРОФІЛЬ КОРИСТУВАЧА, ПРОФІЛЬ ТОВАРУ.

Мета роботи – розробка рекомендаційної системи для вибору фільмів.

Об'єкт дослідження – рекомендаційні системи товарів.

Предмет дослідження – методи та алгоритми формування рекомендацій.

Методи дослідження – методи колаборативної фільтрації, контентні методи та їх поєднання.

У роботі проаналізовано методи формування рекомендацій, проведений огляд існуючих рекомендаційних систем.

Результатом роботи є запропонований власний алгоритм формування рекомендацій на основі дворівневої моделі, яка поєднує підходи контентної та колаборативної фільтрації. Практичним результатом роботи є розробка рекомендаційної системи відео для користувачів онлайн кінотеатру компанії Megogo.

Результати даної роботи рекомендовано використовувати у випадках, коли необхідно сформувати рекомендацію у вигляді списку товарів.

Шляхи подальшого розвитку предмету дослідження – доцільно звернути увагу на проблему “холодного старту” та спосіб оцінки якості моделі шляхом отримання явного відгуку від користувачів.

ABSTRACT

Thesis: 81 p., 27 fig., 7 tabl., 2 appendixes, 21 sources.

RECOMMENDER SYSTEM, COLLABORATIVE FILTERING, CONTENT FILTERING, HYBRID MODEL, USER PROFILE, PRODUCT PROFILE.

The purpose of this work is to implement film recommender system.

The object of the research is the goods recommender system.

The subject of the research is the methods and algorithms for forming recommendations.

Methods of research - methods of collaborative filtering, content methods and their combination.

The paper analyzes the methods of forming recommendations and reviews the existing recommender systems.

The result of the work is the proposed own algorithm for the formation of recommendations based on a two-tier model, which combines the approaches of content and collaborative filtration. The practical result is the development of a recommendation video system for users of the online cinema company Megogo.

The results of this work are recommended to be used in cases when it is necessary to formulate a recommendation in the form of a list of goods.

Ways of further development of the subject of research - it is advisable to pay attention to the problem of "cold start" and how to assess the quality of the model by obtaining explicit feedback from users.

ЗМІСТ

ВСТУП

Не так давно, люди жили в невеликих спільнотах. Кожен продавець знав особисто своїх покупців та міг запропонувати персональні рекомендації грунтуючись на власних знаннях про покупця та його попередні покупки. Такий тип персональних відносин забезпечував для покупців чудовий сервіс, оскільки продавці знали потреби покупців, їхні уподобання, платоспроможність та надавили їм релевантні поради щодо покупки.

Сьогодні, у зв’язку з глобалізацією сервісів, спосіб взаємодії продавців з покупцями суттєво змінився. Хоч споживачі мають можливість вибору різноманітних товарів в інтернет-магазині, вони втратили особистісні зв’язки між продавцем та покупцем та переваги від них. Нестача персоналізованих сервісів не змінила дуже важливий факт, що ключ до успішних продажів - це розуміння особистих проблем кожного покупця [1].

Щоб вирішити цю проблему - зараз активно розробляються та вдосконалюються рекомендаційні сервіси та алгоритми.

Рекомендаційні системи - це набір технік інформаційного фільтрування, які пропонують користувачам потенційно корисні для них товари.

Мета роботи – розробка рекомендаційної системи для вибору фільмів.

Об'єкт дослідження – рекомендаційні системи товарів.

Предмет дослідження – методи та алгоритми формування рекомендацій.

Методи дослідження – методи колаборативної фільтрації, контентні методи та їх поєднання.

У роботі буде проаналізовано методи формування рекомендацій, проведений огляд існуючих рекомендаційних систем.

Результатом роботи є запропонований власний алгоритм формування рекомендацій на основі дворівневої моделі, яка поєднує підходи контентної та колаборативної фільтрації. Практичним результатом роботи є розробка рекомендаційної системи відео для користувачів онлайн кінотеатру компанії Megogo.

Результати даної роботи рекомендовано використовувати у випадках, коли необхідно сформувати рекомендацію у вигляді списку товарів.

Шляхи подальшого розвитку предмету дослідження – доцільно звернути увагу на проблему “холодного старту” та спосіб оцінки якості моделі шляхом отримання явного відгуку від користувачів.

Робота складається з 4 розділів. В першому розділі розглядається актуальність задачі формування рекомендацій, порівнюються існуючі підходи та методи до надання рекомендацій. В другому розділі описується математичні основи формування рекомендацій методами колаборативної фільтрації, а саме пошуку найближчих сусідів та матричною факторизацією, та методами контентної фільтрації, а саме математична модель градієнтного бустингу. Також в цьому розділі розглядається метрики оцінювання та запропонований власний алгоритм рекомендації на основі дворівневої моделі. Третій розділ описує розроблений програмний продукт та демонструє результати розробленого алгоритму на даних онлайн кінотеатру Megogo. Четвертий розділ присвячений економічній частині.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі надання рекомендацій

Рекомендаційні системи займають важливе місце для таких відомих компаній як Amazon.com, YouTube, Netflix, Spotify, LinkedIn, Facebook, TripAdvisor, Last.fm, IMDb, Google та Yandex. Більше того, багато медіа компаній зараз розробляють рекомендаційні системи як частину свого сервісу, яку вони надають своїм користувачам. Компанія Netflix, онлайн провайдер потокового відео, нагородила призом в мільйон доларів команду, яка перша змогла значно покращити продуктивність їхньої рекомендаційної моделі.

Відповідно до досліджень проведених компанією McKinsey, більше 75% контенту, який дивились користувачі компанії Netflix, був саме запропонований їхньою рекомендаційною системою. Система допомагає споживачам знайти цікавий контент, який вони хочуть подивитись, а також допомагає компанії економити на витратах маркетингу [2].

Торгівельний гігант Amazon заявив, що 35% свого доходу він отримав завдяки їхнім рекомендаційним системам. Перш за все, вони показують товари, які часто купують з тими товарами, які у них вже є в кошику. Також вони показують товари, які схожі до тих, які вони щойно переглянули, та товари які є новими версіями до тих, які вони вже купили. Цікаво ще те, що вони вони використовують рекомендаційні системи не лише на сайті, але і в email розсилках, які показують значний приріст у продажах.

Щороку проводиться всесвітньо відома конференція the Association of Computing Machinery Conference Series on Recommender Systems, на якій

науковці діляться новими розробками в галузі рекомендаційних систем, а також відбувається щорічне змагання RecSys Challenge [3].

Поява та зростання інтернет магазинів має значний вплив на поведінку покупців, надаючи їм доступ до значної різноманітності товарів та інформації. Поки ця свобода покупок зробила інтернет комерцію в багато-мільярдну індустрію, вона також ускладнила для покупців вибір товарів, які найбільше відповідають їхнім потребам. Одним із основних методів вирішення проблеми інформаційного перевантаження є рекомендаційні системи, які забезпечують автоматизоване та персоналізоване пропонування товарів для споживачів [4].

1.2 Аналіз існуючих підходів щодо надання рекомендацій

Рекомендації товарів бувають двох типів: персоналізовані та не-персоналізовані. Прикладом не персоналізованих - це рекомендувати найпопулярніші товари, або рекомендувати відштовхуючитсь від бінес завдань.

Серед підходів персоналізованої рекомендації розрізняють *content-based* (базовані на контенті), *collaborative filtering* (колаборативну фільтрацію) та гібридну техніку, яка поєднує два попередні методи.

Загальний принцип *content-based* методів - знайти спільні характеристики товарів, які отримали схвальну оцінку від користувача, а потім рекомендувати для цього користувача нові товари, які також мають ці характеристики. Рекомендаційні системи, які базуються лише на контенті зазвичай мають проблеми аналізу обмеженого вмісту та надмірної спеціалізації. Аналіз обмеженого вмісту настає коли система має обмежену кількість інформації про користувачів та товари. Наприклад, вимога про

конфіденційність приватних даних не дозволяє їх використовувати для аналізу, або докладна інформація про товар є недоступною, дорогою для зібрання, або складною для аналізу - наприклад картинки, музика, тощо. Іншою проблемою є те, що контент товару зазвичай є недостатнім, щоб визначити його якість. Надмірна спеціалізація, з іншого боку, є побічним ефектом того, як рекомендують *content-based* методи нові товари, коли прогнозована оцінка є високою, якщо товар є схожим до того, який сподобався користувачу. Наприклад, в додатку для рекомендацій фільмів, система може рекомендувати користувачу фільми такого ж жанру, або ті які мають однакових акторів з тими фільмами, які користувач вже подивився. Однак система не спроможна порекомендувати фільми, які відрізняються від попередніх і які також були б цікаві користувачу.

Замість того щоб залежати від контентної інформації, методи колаборативної фільтрації використовують інформацію про оцінки від інших користувачів про товари в системі. Ключовою ідеєю є те, що оцінка певного юзера для нового товару буде схожою з тою, яку поставив інший користувач, якщо вони обидва оцінили інші товари подібними оцінками. Колаборативна фільтрація змогла подолати певні обмеження, які має *content-based.* Наприклад, товари, контент яких є відсутнім або важко доступним, можуть бути рекомендовані базуючись на відгуках інших користувачів. Більше того, колаборативні рекомендації базуються на якості товарів які оцінені іншими, ніж покладатись на контент який може бути поганим індикатором якості товару. Також колаборативна фільтрація може рекомендувати різноманітні товари з різним контентом, оскільки інші користувачі проявили інтерес до цих різних товарів.

Підходи колаборативної фільтрації можуть бути згруповані на два загальні класи базовані *на знаходження сусідів* та базовані на *навчанні*

моделі. В підході основою якого є знаходження сусідів, оцінки користувача для товарів, які збережені в системі, безпосередньо використовуються під час прогнозування оцінок для нових товарів. Це може бути здійснено двома способами відомими як *user-based* та *item-based* рекомендаціями. *User-based* системи оцінюють інтерес певного користувача для товару використовуючи оцінки для цих товарів від інших користувачів, які називають сусідами та мають схожі моделі поведінки оцінювання. Сусіди певного користувача є зазвичай користувачі, оцінки яких найбільше корелюють з оцінками даного користувача. *Item-based* підхід, з іншого боку, прогнозує оцінки користувача для товару базуючись на оцінках користувача для схожих товарів. В таких підходах, два товари є схожими якщо декілька користувачів системи оцінили ці товари подібними оцінками [5].

В противагу методам на знаходження сусідів, які використовують збережені оцінки безпосередньо під час прогнозування, методи базовані на моделях використовують оцінки щоб навчити рекомендувати модель. Важливі характеристики користувачів та товарів зберігаються в множині параметрів моделі, які отримані під час навчання моделі на тренувальних даних і можуть бути використані для прогнозування нових оцінок. Існують різноманітні підходи до створення моделей для задачі рекомендацій. Деякі з них розглянемо детальніше пізніше.

На завершення, щоб подолати певні обмеження методів базованих на контенті та колаборативної фільтрації, використовують гібридні рекомендаційні методи, які поєднують кращі характеристики обох підходів. Поєднати ці методи можна різноманітними способами, наприклад, поєднавши окремі списки рекомендацій в один, або додавання контентної інформації в моделі колаборативної фільтрації. Декілька

досліджень показали, що використання гібридних моделей забезпечило більш точні рекомендації ніж чисті content-based та колаборативні методи, особливо коли наявні лише декілька оцінок.

Оскільки електронна комерція (e-commerce) набирає популярності, важливим завданням є допомогти користувачам легко відсортувати серед великої різноманітності запропонованих товарів ті, які найбільше відповідають їхнім бажанням. Одним з засобів вирішення поставленого завдання є рекомендаційні системи, які зараз активно розробляються та вдосконалюються. Ці системи забезпечують користувачів персоналізованими рекомендаціями товарів або сервісами, які бажано б підходили їхнім смакам та потребам. Технології, які використовуються, базуються на створенні профайлу користувачів та товарів та знаходженню як їх взаємопов’язати.

Рекомендаційні системи базуються на двох різних стратегіях (або поєднанню них). Підхід базований на контенті (Content based) створює профіль для кожного користувача та продукту з їхніх характеристик. Наприклад, профіль фільму може містити ознаки жанру, акторів, касової популярності, тощо. Профіль користувача може містити демографічну інформацію або відповіді на запропоновані запитання. Отримані профайли надають змогу програмам асоціювати користувачів з відповідними товарами. Однак, цей підхід потребує збирання інформації, яка може не бути в наявності або складною для накопичення.

Альтернативна стратегія потребує лише історії поведінки користувача без створення явного профілю. Такий підхід відомий як Collaborative Filtering (CF) суспільна фільтрація.

CF аналізує взаємозв’язки між користувачами та залежності серед товарів, щоб знайти нові користувач-товар асоціації. Наприклад, деякі CF системи

знаходять групи товарів завдяки тому що вони були оцінені однаково або вподобані юзерами з схожою історією оцінок.

1.3 Особливості предметної області

Оцінки вподобань користувачем товарів можуть бути числові значення відомі як рейтинги (наприклад зірочки 1-5), бінарні значення (сподобалось/не сподобалось). Більше того, відгуки користувача можуть бути отримані в явному (explicit) вигляді, наприклад введені користувачем в систему, або неявні (implicit), отримані з історії покупок, переглядів, тощо.

Оцінки отримані у явному вигляді - вважаються більш цінними, оскільки користувач особисто вказав, наскільки йому подобається даний товар. З іншого боку, користувачі часто не залишають відгуки - це призводить до втрати важливої інформації, яка могла б покращити рекомендації. Тому з неявних відгуків можна отримати значно більше інформації про вподобання того чи іншого користувача.

Першим чинником, який слід врахувати при розробці РС, є область застосування, оскільки вона має значний вплив на алгоритмічний підхід, який необхідно прийняти. Монтанер в [6] надає таксономію РС та класифікує існуючі додатки РС до певних доменів додатків. Виходячи з цих конкретних доменів додатків, розглянемо більш загальні класи доменів для найпоширеніших додатків рекомендаційних систем:

- Розваги - рекомендації для фільмів, музики, ігор, тощо.
- Контент - персоналізовані газети, новини, рекомендації документів, веб сторінок в пошукових запитах, рекомендації онлайн курсів, а також фільтрація по категоріям для електронної пошти.

- Електронна комерція — рекомендації товарів для покупок, таких як книжки, телефони, ноутбуки, аксесуари, тощо.
- Послуги - рекомендації туристичних послуг, рекомендації експертів для консультації, рекомендації будинків для оренди або послуги знайомств.
- Соціальні мережі - рекомендація людей у соціальних мережах та рекомендації контенту в соціальних медіа, такі як твіти, канали Facebook, оновлення LinkedIn та інші.

Основні функції рекомендаційної системи:

Збільшити кількість проданих товарів. Це напевно найважливіша функція комерційних рекомендаційних систем - можливість продати додатковий набір товарів в порівнянні з тим випадком, коли під час продажів не використовувати будь-який з видів рекомендації. Ця мета досяжна, оскільки рекомендовані товари зазвичай задовольняють потребам та смакам користувачів. Можливо користувач знайде цікавий йому товар після запропонованих декількох рекомендацій. Некомерційні організації мають подібні цілі, навіть тоді, коли для користувача нічого не коштує, те що він вибрав додаткові товари. Наприклад, новинний портал хоче збільшити кількість переглянутих новин на сайті. Загалом, головною метою впровадження рекомендаційної системи є збільшення відсотку конверсії. Це можна побачити, порівнявши кількість користувачів які прийняли рекомендацію та спожили товар, в порівнянні до кількості простих відвідувачів, які просто шукали товари серед всієї інформації.

Продавати більш різноманітні товари. Іншою основною функцією рекомендаційних систем є надання користувачу можливості вибрати товари, які важко було б знайти без рекомендації. Наприклад, туристичні

компанії зацікавлені в тому, щоб рекомендаційна система рекомендувала відвідувачам всі можливі туристичні місця, а не лише найпопулярніші. Це може бути досить складно без рекомендаційних систем, оскільки компанія не може ризикувати рекламувати ті місця, які можуть не сподобатися користувачу. Саме тому, рекомендаційні системи пропонують або рекламують не дуже популярні місця саме для тих користувачів, яким потенційно вони можуть сподобатись.

Збільшити задоволення користувача. Добре розроблена рекомендаційна система може покращити враження користувача від користування сайтом або додатком. Якщо рекомендовані товари будуть корисними, релевантними, та розроблена система буде мати зручний та привабливий вигляд, то користувачі будуть отримувати задоволення від використання системи. Поєднання ефективних, точних рекомендацій та приємного користувацького інтерфейсу значно підвищать особисте ставлення користувача до системи. Це, в свою чергу, збільшить корисність системи та більш ймовірно, що рекомендації будуть прийняті та спожиті.

Збільшити лояльність користувачів. Користувачі будуть більш відданим до веб сервісу, якщо під час відвідування, система розпізнає їх як минулих покупців та буде поводиться з ними як з цінними відвідувачами. Це є стандартна особливість рекомендаційних систем, оскільки вони здійснюють рекомендації, на основі попередніх взаємодій користувача, такі як переглянуті, куплені товари, та залишені до них оцінки.
Відповідно, чим довше користувач взаємодіє з веб сайтом, тим більш модель стає адаптованою до потреб користувача, отримує краще розуміння його уподобань, рекомендації стають більш персоналізовані.

Краще розуміння потреб користувача. Іншою важливою функцією рекомендаційної системи, яка може бути використана в багатьох інших

програмах - це опис уподобань користувача, який збирається від користувачів у явному вигляді або прогнозується системою. Постачальник послуг може потім вирішити перевикористати ці знання для багатьох інших цілей, таких як покращити управління запасами товарів та виробництва. Наприклад, в контексті туристичного бізнесу, управління напрямками в організації може вирішити рекламувати специфічні регіони для певних груп користувачів базуючись на аналізі інформації зібраної рекомендаційною системою [7].

1.4 Постановка задачі дослідження

- Провести дослідження існуючих методів формування рекомендацій
- Розробити власний алгоритму формування рекомендацій на основі дворівневої моделі
- Провести оцінку якості роботи запропонованого алгоритму
- Програмно реалізувати рекомендаційну системи на основі розробленого алгоритму

1.5 Висновки до розділу 1

В розділі 1 було проаналізовано актуальність задачі формування рекомендацій товарів, виявлено проблеми та можливі напрямки досліджень в цій галузі. Було розглянуто особливості рекомендаційних систем, їхні переваги для впровадження в бізнесі.

Також було проаналізовано існуючі підходи до формування рекомендацій. На підставі розглянутих підходів можна виділити 3 основних напрямки:

- рекомендаційні системи на основі вмісту;
- рекомендаційні системи на основі колаборативної фільтрації;
- гібридні системи, які поєднують 2 попередні підходи.

Було сформовано постановку завдання.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Дослідження існуючих методів надання рекомендацій

2.1.1 Методи колаборативної фільтрації

2.1.1.1 Методи на основі пошуку найближчих сусідів

User-based підхід найближчих сусідів для рекомендації оцінки r_{ui} користувачем u для нового товару i використовує оцінки товара i які дали найбільш схожі на u користувачі. Для кожного користувача $v \neq u$ значення w_{uv} означає міру схожості між користувачем u та v (деякі міри схожості будуть описані пізніше). K найближчих сусідів до користувача u є k користувачів v , які мають найбільшу міру схожості w_{uv} до u, їх множину позначають $N(u)$. Однак, лише ті користувачі, які оцінили товар i можуть бути використані для прогнозу r_{ui}, тому розглядають k користувачів схожих на u які оцінили товар i, їх позначають $N_i(u)$ [8].

Для прогнозу оцінки користувачем u для товару i використовують зважене середнє оцінок товару i, які поставили найближчі сусіди з $N_i(u)$:

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} r_{vi}}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}.$$

В знаменнику формули, ми використовуємо $|w_{uv}|$ замість w_{uv}, оскільки міра схожості може набувати і від’ємних значень.

Щоб порівняти чи прогнозована оцінка є кращою чи гіршою ніж середнє значення оцінки товару, використовують центрування оцінки

відносно середнього. В такому разі можна легко зрозуміти, чи даний товар сподобається користувачу чи ні.

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} (r_{vi} - \bar{r}_v)}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}.$$

Досить часто щоб покращити якість прогнозу оцінки товару використовують нормалізацію [9]. Позначимо σ_u - стандартне відхилення оцінок користувача u, а σ_v - стандартне відхилення оцінок найближчого сусіда v користувача u. Тоді $\hat{r}_{ui}$ обчислюють за такою формулою:

$$\hat{r}_{ui} = \bar{r}_u + \sigma_u \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} (r_{vi} - \bar{r}_v)/\sigma_v}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}.$$

В підході user-based ми покладаємось на думку однодумців користувачів щоб спрогнозувати оцінку. В підході item-based ми дивимось на оцінки, які поставили подібним товарам. Позначимо $N_u(i)$ - множина товарів, яку оцінив користувач u, які найбільш схожі до товару i. Прогнозована оцінка користувачем u товару i обчислюється як зважене середнє оцінок поставлених користувачем u для товарів $N_u(i)$:

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij} r_{uj}}{\sum\limits_{j \in \mathcal{N}_u(i)} |w_{ij}|}.$$

Для обчислення оцінки $\hat{r}_{ui}$ з центруванням відносно середньої оцінки $\bar{r}_i$ товару i використовують формулу:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} (r_{uj} - \bar{r}_j)}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}.$$

Для обчислення $\hat{r}_{ui}$ з нормалізацією використовують формулу:

$$\hat{r}_{ui} = \bar{r}_i + \sigma_i \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} (r_{uj} - \bar{r}_j)/\sigma_j}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}.$$

де σ_i - стандартне відхилення оцінок товару i;

σ_j - стандартне відхилення оцінок найближчого сусіда j до товару .

Міра схожості відіграє важливу у роль в методах найближчих сусідів [10]:

- Вона дозволяє вибрати найбільш подібних сусідів, чиї оцінки будуть використовуватись під час прогнозу рекомендованої оцінки
- Вона впливає на те, наскільки сильно враховувати вплив на оцінку від того чи іншого найближчого сусіда.

Обчислення вагів міри схожості має значний вплив на точність та продуктивність для методів найближчих сусідів [11].

Нехай r_{ui} - оцінка яку поставив користувач u для товару i.

r_{vi} - оцінка яку поставив користувач v для товару i відповідно.

$\bar{r}_u$ - середнє арифметичне оцінок товарів, які ставив користувач u

$\overline{r}_v$ - середнє арифметичне оцінок товарів, які ставив користувач v

I_u - множина товарів, які оцінив користувач u

I_v - множина товарів, які оцінив користувач v

I_{uv} - множина товарів, які оцінили обидва користувачі u та v

$|I_u|$ - кількість товарів, які оцінив користувач u

$|I_v|$ - кількість товарів, які оцінив користувач v

$|I_{uv}|$ - кількість товарів, які оцінили обидва користувачі u та v

Косинусна відстань

$$CV(u,v) = \frac{\sum\limits_{i\in\mathcal{I}_{uv}} r_{ui}\, r_{vi}}{\sqrt{\sum\limits_{i\in\mathcal{I}_u} r_{ui}^2 \sum\limits_{j\in\mathcal{I}_v} r_{vj}^2}},$$

Кореляція Пірсона

$$\mathrm{PC}(u,v) = \frac{\sum\limits_{i\in\mathcal{I}_{uv}} (r_{ui}-\overline{r}_u)(r_{vi}-\overline{r}_v)}{\sqrt{\sum\limits_{i\in\mathcal{I}_{uv}} (r_{ui}-\overline{r}_u)^2 \sum\limits_{i\in\mathcal{I}_{uv}} (r_{vi}-\overline{r}_v)^2}}.$$

Коефіцієнт Жаккара

$$J(u,v) = \frac{|I_{uv}|}{|I_u|+|I_v|-|I_{uv}|}$$

2.1.1.2 Методи базовані на матричній факторизації

Моделі з латентними факторами намагаються виявити приховані характеристики які пояснюють спостережувані оцінки. Такий підхід намагається апроксимувати матрицю з відомими значеннями оцінок добутком двох матриць, кожна з яких складається з прихованих характеристик користувачів та товарів відповідно [12] (рис. 2.1).

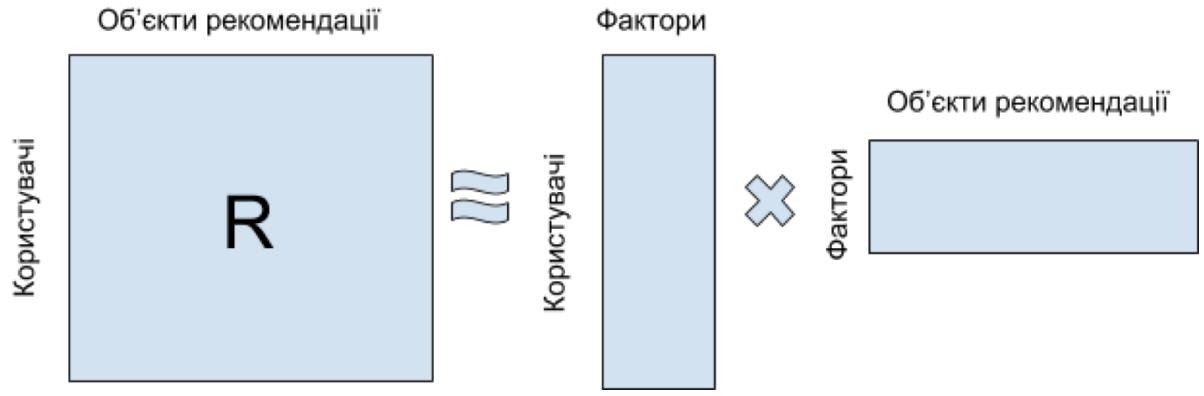


Рисунок 2.1 – Факторизація матриці рейтингів

Розглянемо Baseline модель. Позначимо μ - середня значення оцінки для тренувальних даних

Тоді прогнозоване значення оцінки користувачем u для товару i знаходимо за формулою:

$$\hat{r}_{ui} = \mu + b_u + b_i$$

Параметри b_u та b_i вказують на спостережуване відхилення від середнього значення оцінок для користувача u та товару i відповідно.

Щоб знайти значення b_u та b_i оптимізують функцію втрат:

$$\min_{b*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2) .$$

Константа λ_1 - відповідає за регуляризацію, щоб модель не перевчилась.

Для оптимізації функції втрат можна використати метод стохастичного градієнтного спуску.

Роглянемо SVD модель [13]. Моделі матричної факторизації представляють користувачів та товари в просторі латентних змінних розмірності f. Кожен товар i представлений вектором $q_i \in R^f$, кожен користувач u представлений вектором $p_u \in R^f$.

Для певного товару i, значення q_i означають міру певних властивостей, якими характеризують товари. Відповідно для користувача u, значення p_u означають міру певних властивостей, якими зацікавлені користувачі в товарах. Значення у векторах можуть бути як позитивними, так і негативними. Скалярний добуток $q_i^T p_u$ характеризує зацікавленість користувача u характеристиками товару i.

Остаточна оцінка отримується додаванням відомих з попередньої моделі параметрів, які залежать лише від користувача та товарів.

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u .$$

Щоб знайти параметри μ, b_u, b_i, q_i та p_u оптимізуємо функцію втрат:

$$\min_{b_*, q_*, p_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda_4 (b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2) .$$

Константа λ_4, яка відповідає за міру регуляризації, зазвичай підбирається на кросс-валідації. Оптимізація зазвичай здійснюється методом стохастичного градієнтного спуску [14].

Розглянемо алгоритм implicit ALS [15]. Для початку, введемо поняття впевненості вподобання користувачем u товару i, яке позначимо змінною r_{ui}. Введемо множину бінарних змінних p_{ui}, які вказують наявність вподобання між користувачем та товаром. Змінні p_{ui} утворені бінаризацією значень r_{ui}:

$$p_{ui} = \begin{cases} 1, & r_{ui} > 0; \\ 0, & r_{ui} = 0; \end{cases}$$

Якщо користувач u мав взаємодію з товаром i ($r_{ui} > 0$), то маємо індикатор $p_{ui} = 1$. В іншому випадку, якщо u ніколи не взаємодіяв з i, то $p_{ui} = 0$. Природа значень p_{ui} з нульовими значеннями пов'язана з недостатньою впевненістю в тому, чи користувач не любить цей товар, чи просто по якимось причинам він не взаємодіяв з ним. Наприклад, користувач не знав про існування такого товару, або не мав можливості придбати через його вартість, або обмеженість в наявності, тощо. Також, споживання товару могло стати за іншої причини ніж вподобання його. Наприклад, користувач подивився телевізійне шоу лише через те, що він не перемкнув канал після перегляду попереднього шоу. Або користувач купив товар в якості подарунку для когось, а не для себе та немає потреби в ньому. Тому існують різні рівні впевненості у вподобанні серед товарів які

мали взаємодію з користувачем. Зі збільшенням r_{ui} зростає впевненість в тому, що користувачу дійсно подобається даний товар. Саме тому, вводиться набір змінних c_{ui}, які характеризують міру довіри під час спостереженні p_{ui}.

$$c_{ui} = 1 + \alpha r_{ui}$$

В такому випадку, ми маємо незначну впевненість для кожної пари користувач-товар, однак, оскільки ми спостерігаємо більше доказів для позитивних вподобань, наша впевненість в $p_{ui} = 1$ зростає відповідно. Швидкість зростання контролюється змінною α.

Метою є знайти вектор $x_u \in R^f$ для кожного користувача u, вектор $y_i \in R^f$ для кожного товару i, що буде відображати вподобання користувача. Іншими словами, вподобання є скалярним добутком фактор векторів користувача та товару відповідно.

$$p_{ui} = x_u^T y_i$$

Вектори намагаються представити користувачів та товари через прихований факторний простір, у якому їх можна безпосередньо порівняти. Це схоже на техніки матричної факторизації, які популярні для даних з явними залишеними відгуками, з двома суттєвими відмінностями:

- Потрібно враховувати змінний рівень впевненості
- Оптимізація повинна здійснюватись для всіх можливих пар u, i, а не тільки для тих, які є у досліджуваних даних. Відповідно, фактори обчислюються при мінімізації наступної функції втрат:

$$min_{x_*, y_*} \Sigma_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\Sigma_u \|x_u\|^2 + \Sigma_i \|y_i\|^2)$$

Вираз $\lambda(\Sigma_u \|x_u\|^2 + \Sigma_i \|y_i\|^2)$ необхідний для регуляризації моделі, щоб вона не пере навчилась під тренувальні дані. Точне значення параметра λ є залежним від даних, тому воно визначається під час кросс-валідації.

Зауважимо, що функція втрат містить $m \cdot n$ доданків, де m - це кількість користувачів, n - кількість товарів. Для типових наборів даних число $m \cdot n$ може з легкістю досягти декількох мільярдів. Ця велика кількість доданків перешкоджає найбільш прямій оптимізаційній техніці таку як стохастичний градієнтний спуск, який широко застосовується для наборів даних з явними відгуками. Тому, запропоновано альтернативний ефективний оптимізаційний метод, про який буде йти мова далі.

Зазначимо, що коли або фактори користувача, або фактори-товарів зафіксовані, функція втрат стає квадратичною, тому її глобальний мінімум легко обчислюється. Це призводить до навперемінного методу найменших квадратів, коли почергово переобчислюються то фактори користувача, то фактори товарів. Кожен крок гарантує зменшення значення функції втрат.

Для початку переобчислимо всі фактори користувачів. Припустимо, що всі фактори товарів зібрані в матрицю Y розмірності $n \times f$. Перед тим як проходитись в циклі по всім користувачам, обчислюємо матрицю $Y^T Y$ розмірності $f \times f$ за час $O(f^2 n)$. Для кожного користувача u, визначимо діагональну матрицю C^u розмірності $n \times n$, де $C^u_{ii} = c_{ui}$, та також вектор $p(u) \in R^n$, який містить всі вподобання користувачем u (значення p_{ui}). За допомогою розмежування знаходимо аналітичний вираз для x_u, який мінімізує функцію втрат:

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

Складність полягає в обчисленні $Y^T C^u Y$, яке потребує часу $O(f^2 n)$ для кожного з m користувачів. Значне пришвидшення досягається використанням факту, що $Y^T C^u Y = Y^T Y + Y^T (C^u - I) Y$. В такому разі, $Y^T Y$ є незалежним від u та вже є завчасно обчисленим. Для $Y^T (C^u - I) Y$ зауважимо, що $C^u - I$ має лише n_u ненульових компонент, де n_u - кількість товарів для яких $r_{ui} > 0$ і зазвичай $n_u << n$. За схожим принципом, $C^u p(u)$ містить лише n_u ненульових компонент. Через це, переобчислення x_u виконується за час $O(f^2 n_u + f^3)$. Тут припускається, що $O(f^3)$ - час знаходження оберненої матриці $(Y^T C^u Y + \lambda I)^{-1}$. Цей крок виконується для кожного з m користувачів, тому загальний час обчислення буде $O(f^2 N + f^3 m)$, де N - сумарна кількість ненульових спостережень, тобто $N = \Sigma_u n_u$. Важливо, час обчислень лінійно пропорційний до вхідних даних. Зазвичай значення f лежить між 20 та 200.

Переобчислення факторів користувача слідує за переобчисленням всіх факторів товарів, які обчислюються розпаралелено. Спочатку впорядковуються всі фактори користувачів в матрицю X розміром $m \times f$. Обчислюється матриця $X^T X$ розмірності $f \times f$ за час $O(f^2 m)$. Для кожного товару i визначається діагональна матриця C^i розмірності $m \times m$, де $C^i_{uu} = c_{ui}$, а також вектор $p(i) \in R^m$ який містить всі вподобання для товару i. Далі ми обчислюємо:

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i)$$

Використовуючи ту саму техніку як і для факторів користувача, час обчислення буде $O(f^2 N + f^3 m)$. Загалом виконується декілька ітерацій

попарних переобчислень факторів користувачів та товарів, допоки вони не стабілізуються. Типовим числом ітерацій є 10.

Весь процес маштабується лінійно в залежності від розміру вхідних даних. Після обчислення факторів користувача та товарів, алгоритм рекомендує для u користувача K наявних товарів з найбільшим значенням $\hat{p_{ui}} = x_u^T y_i$, де $\hat{p}_{ui}$ означає прогнозоване вподобання користувача u до товару i.

2.1.2 Методи на основі контенту

2.1.2.1 Лінійні моделі

Задачу прогнозу оцінки товару можна розглядати як задачу регресії за наявності інформації, яка характеризує товари та користувачів [16].

Нехай

X - простір ознак об’єктів

Y - простір відповідей

$x = (x^1, .., x^d)$ - ознаки які характеризують об’єкт

$X = (x_i, y_i)_{i=1}^l$ - навчальна вибірка

$a(x)$ - модель алгоритму

$Q(a, X)$ - функціонал похибки алгоритму a на вибірці X

A - множина моделей алгоритму, серед яких вибирають найкращий

Процес навчання: $a(x) = argmin_{a \in A} Q(a, X)$

Модель алгоритму лінійної регресії виглядає наступним чином:

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x^j$$

де w_0 - вільний коефіцієнт

x^j - ознаки об’єкту

w_j - вага ознаки

Якщо додати $(d+1)$-шу ознаку, яка приймає значення 1, то можна записати у вигляді:

$$a(x) = \sum_{j=1}^{d+1} w_j x^j = \langle w, x \rangle$$

Щоб знайти ваги w_j під навчання моделі, оптимізовують функціонал середньо квадратичної похибки:

$$Q(w, x) = \frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2$$

Також в задачі рекомендації товарів можна розглядати підхід бінарної класифікації об’єкта з класами “сподобається користувачу” та “не подобається користувачу”. В такому підході ми обраховуємо ймовірність того, що товар сподобається користувачу, а потім рекомендуємо ті товари, які мають найбільшу ймовірність.

В задачі класифікації можна використовувати сигмоїду (рис.2.1):

$$\pi(x) \approx \frac{e^{\langle w,x \rangle}}{1 + e^{\langle w,x \rangle}}$$

$$\pi(x) \in [0, 1]$$

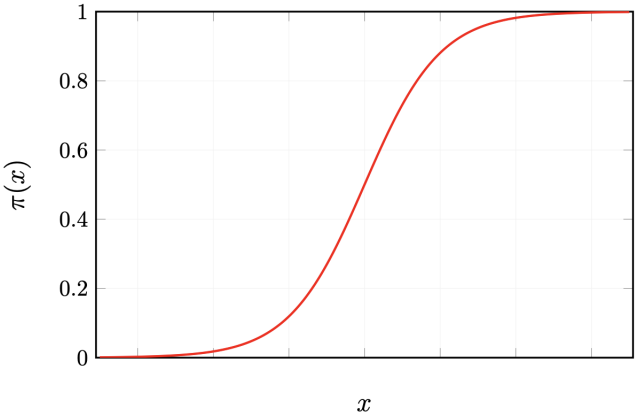


Рисунок. 2.2 Графік функції сигмоїди

Щоб знайти параметри моделі, спочатку виразимо

$$\langle w, x \rangle \approx \ln \frac{\pi(x)}{1 - \pi(x)}$$

Налаштування моделі відбувається максимізацією правдоподібності. Але зручніше не максимізувати правдоподібність, а мінімізувати мінус логарифм від правдоподібності:

$$-\ln L(X) = -\sum_{i=1}^{\ell} \Big(y_i \ln \pi(x_i) + (1 - y_i) \ln(1 - \pi(x_i)) \Big)$$

2.1.2.2 Нелінійні моделі

Лінійні моделі можуть виявляти лише прості лінійні залежності. У випадку взаємозалежних ознак об’єкта та складних залежностей часто використовують нелінійні алгоритми на основі дерев рішень [17].

Дерева рішень зазвичай представляють собою бінарні дерева, в яких в кожній внутрішній вершині записана умова, а в кожному листі дерева - прогноз. Загалом можуть бути і не бінарні дерева, але зазвичай використовують саме їх.

Умови в внутрішніх вершинах вибирають досить простими. Зазвичай це перевірити, чи знаходиться значення певної ознаки x^j ліворуч від заданого порогового значення t:

$$[x^j \leq t].$$

Це досить проста умова, яка залежить лише від однієї ознаки, але її достатньо, щоб розв'язувати різні складні задачі.

В машинному навчанні використовується жадібний алгоритм побудови розв'язуючого дерева від кореня до листка. На початку вибирається корінь, який розбиває вибірку на дві частини. Потім розбивається кожен з нащадків цього кореня і так далі. Дерева розгалужуються до тих пір, поки це необхідно.

Нехай в вершину m потрапила множина X_m об'єктів з навчальної вибірки. Параметри в умові $[x^j \leq t]$ будуть вибрані таким чином, щоб мінімізувати критерій похибки, який залежить від цих параметрів:

$$Q(X_m, j, t) \to \min_{j,t}$$

Після того, як параметри були вибрані, множина X_m об'єктів з навчальної вибірки розбивається на дві множини

$$X_\ell = \left\{x \in X_m | [x^j \leq t]\right\}, \qquad X_r = \left\{x \in X_m | [x^j > t]\right\}$$

кожен з яких відповідає своїй дочірній вершині. Запропоновану процедуру можна продовжити для кожної з дочірніх вершин - тоді дерево буде все більше та більше заглиблюватись. Такий процес з часом має закінчитись і наступна дочірня вершина має бути оголошена листком, а не розділена навпіл. Цей момент визначається критерієм зупинки. Існує багато різних критеріїв:

- Якщо в вершину потрапив лише один об’єкт навчальної вибірки, або всі об’єкти належать одному класу (в задачі класифікації).
- Можна зупинити розбиття, якщо глибина дерева досягла певного значення.

Коли вершину оголосили листом, потрібно визначити прогноз, який буде знаходитись в даному листі. В даний лист потрапила під вибірка X_m навчальної вибірки, і потрібно вибрати прогноз, який буде оптимальний для даної підвибірки. В задачі класифікації оптимально повертати той клас, який найбільш часто зустрічається серед об’єктів в X_m :

$$a_m = \text{argmax}_{y \in \mathbb{Y}} \sum_{i \in X_m} [y_i = y]$$

Якщо потрібно вказати ймовірності класів, їх можна вказати як долю об’єктів різних класів в X_m :

$$a_{mk} = \frac{1}{|X_m|} \sum_{i \in X_m} [y_i = k]$$

Критерій похибки записується наступним чином:

$$Q(X_m, j, t) = \frac{|X_\ell|}{|X_m|} H(X_\ell) + \frac{|X_r|}{|X_m|} H(X_r)$$

Функція $H(X)$ - називається критерієм інформативності. Його значення повинно бути тим менше, чим менший розкид відповідей в X.

Для задачі класифікації зазвичай використовують критерій Джині:

$$H(X) = \sum_{k=1}^{K} p_k (1 - p_k)$$

$$p_k = \frac{1}{X} \sum_{i \in X} [y_i = k]$$

де p_k - доля об'єктів класу k у вибірці X.

Дерева рішень мають певні недоліки:

- сильно пере навчаються
- сильно змінюються при невеликій зміні вибірки

Ці недоліки можна перетворити в переваги, використовуючи композицію моделей.

Композиція - це об'єднання N алгоритмів $b_1(x), ..., b_N(x)$ в один. Ідея полягає в тому, що спочатку навчаєш алгоритми $b_1(x), ..., b_N(x)$, а потім усереднюєш значення отриманих від них відповідей:

$$a(x) = \frac{1}{N} \sum_{n=1}^{N} b_n(x)$$

Бустинг - це один із способів побудови композицій, під час якого:

- Базові алгоритми будуються послідовно один за одним
- Кожен наступний алгоритм будується таким чином, щоб виправити помилки вже побудованої композиції.

Градієнтний бустинг є одним із кращих способів направленої побудови композицій [18]. В ньому побудована композиція є сумою, а не усередненням базових алгоритмів:

$$a_N(x) = \sum_{n=1}^{N} b_n(x)$$

Це пов’язано з тим, що алгоритми навчаються послідовно і кожен наступний коректує помилку попередніх.

Нехай задана функція втрат $L(y, z)$, де y - справжнє значення об’єкта, z - прогнозоване значення алгоритму.

Прикладом функції втрат для задачі класифікації може бути логістична функція втрат:

$$L(y, z) = \log(1 + \exp(-yz))$$

2.2 Критерії якості для задачі надання рекомендацій

Метрики якості прогнозу оцінки

Нехай N - загальна кількість всіх оцінок, які використовують для обчислення метрики, r_{ui} - істинне значення оцінки, яку поставив користувач u для товару i, $\hat{r}_{ui}$ - прогнозоване значення оцінки

Mean Square Error (MSE)

$$MSE = \frac{1}{N}\sum_{ui}(r_{ui} - \hat{r}_{ui})^2$$

Root Mean Square Error (RMSE)

$$RMSE = \frac{1}{N}\sum_{ui}\sqrt{(r_{ui} - \hat{r}_{ui})^2}$$

Mean Absolute Error (MAE)

$$MAE = \frac{1}{N}\sum_{ui}|r_{ui} - \hat{r}_{ui}|$$

Розглянемо метрики якості рекомендацій. Розглянемо випадок, коли для кожного користувача u система повертає впорядкований список з K товарів [19].

Нехай $Relevance(k)$ - індикатор релевантності рекомендованого товару на k-тій позиції, який може набувати значення $\{0,1\}$.

Релевантність товару може визначатись:

- За історичними даними - користувачу в майбутньому сподобався рекомендований товар
- За експертною оцінкою - самостійно зазначити які товари вважати релевантними

Precision at K

$$Precision@K = \frac{1}{K} \sum_{k=1}^{K} Relevance(k)$$

Метрика вказує скільки серед рекомендованих товарів було релевантних для одного користувача. Недоліком є те, що не враховує порядок розташування релевантних товарів.

Average precision at K

$$Average\ Precision@K = \frac{1}{K} \sum_{k=1}^{K} Relevance(k) \cdot (Precision@k)$$

Щоб порахувати значення метрики для N користувачів:

$$MAP@K = \frac{1}{N} \sum_{j=1}^{N} Average\ Precision@K_j$$

Discounted Cumulative Gain at K

$$DCG@K = \sum_{k=1}^{K} \frac{Relevance(k)}{log_2(k+1)}$$

Normalized Discounted Cumulative Gain at K

$$NDCG@K = \frac{DCG@K}{max(DCG@K)}$$

де $max(DCG@K)$ - це максимальне значення, яке може набувати $DCG@K$

Щоб порахувати значення метрики для N користувачів:

$$m\,NDCG@K = \frac{1}{N}\sum_{j=1}^{N} NDCG@K_j$$

Mean reciprocal rank

$$MRR@K = \frac{1}{N}\sum_{j=1}^{N} RR@K_j$$

$$RR@K = \frac{1}{min\{k \in [1..K] : r(k) = 1\}}$$

2.3 Алгоритм розв'язку задачі, що пропонується в роботі

Розглянемо розроблену дворівневу модель рекомендації товарів, яка використовує метод матричної факторизації implicit als на першому рівні та алгоритм градієнтного бустингу cat boost на другому [20].

Нехай U - множина користувачів системи, I - множина товарів, які може запропонувати система.

Як вже зазначалось вище, введемо поняття впевненості вподобання користувачем $u \in U$ для товару $i \in I$, яке позначимо змінною r_{ui}. Воно може набувати як цілих, так і неперервних невід'ємних значень, тобто $r_{ui} \in [0;\ +\infty)$. Введемо множину бінарних змінних p_{ui}, які вказують

наявність вподобання між користувачем та товаром. Якщо користувач u мав взаємодію з товаром i ($r_{ui} > 0$), то маємо індикатор $p_{ui} = 1$. В іншому випадку, якщо u ніколи не взаємодіяв з i, то $p_{ui} = 0$. Природа значень p_{ui} з нульовими значеннями пов'язана з недостатньою впевненістю в тому, чи користувач не любить цей товар, чи просто по якимось причинам він не взаємодіяв з ним. Зі збільшенням r_{ui} зростає впевненість в тому, що користувачу дійсно подобається даний товар. Саме тому, вводиться набір змінних c_{ui}, які характеризують міру довіри під час спостереженні p_{ui}:

$$c_{ui} = 1 + \alpha r_{ui}.$$

В такому випадку, ми маємо незначну впевненість для кожної пари користувач-товар, однак, оскільки ми спостерігаємо більше доказів для позитивних вподобань, наша впевненість в $p_{ui} = 1$ зростає відповідно. Швидкість зростання контролюється змінною α.

Метою 1 рівня є знайти вектор $x_u \in R^f$ для кожного користувача u, що буде відображати вподобання користувача, та вектор $y_i \in R^f$ для кожного товару i, що характеризує товар. Міра вподобання є скалярним добутком фактор векторів користувача та товару відповідно:

$$p_{ui} = x_u^T y_i$$

Вектори намагаються представити користувачів та товари через прихований факторний простір, у якому їх можна безпосередньо порівняти. Фактори обчислюються при мінімізації наступної функції втрат:

$$min_{x_*, y_*} \Sigma_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\Sigma_u \|x_u\|^2 + \Sigma_i \|y_i\|^2)$$

Знайшовши фактор вектори вподобань, знаходимо для кожної пари користувача u та товару i значення $\hat{p}_{ui} = x_u^T y_i$, яке прогнозує на скільки товар може сподобатись користувачу. Далі для кожного користувача u вибираємо K_1 кількість товарів, для яких маємо найбільше значення $\hat{p}_{ui}$. Ці товари відібрані як кандидати для рекомендації для наступної більш складної моделі, яка відсортує та вибере з них остаточні K_2 товарів, які будуть зарекомендовані користувачу. Позначимо множину кандидатів для кожного користувача як I_u.

Отже, від моделі на 1 рівні для кожного товару i отримали вектор $y_i \in R^f$, що характеризує товар, для кожного користувача u отримали вектор $x_u \in R^f$, який описує вподобання користувача, а також набір кандидатів для рекомендації I_u.

На 2 рівні ми використовуємо більше складну модель градієнтного бустингу - CatBoost.

Спочатку введемо порогове значення t, яке поділить навчальну вибірку вибірку на 2 класи {“товар сподобався”, “товар не сподобався”}. Якщо міра вподобання $r_{ui} > t$ для користувача u та товару i, то відносимо цю взаємодію до класу “товар сподобався”, якщо $r_{ui} < t$ - то “товар не сподобався”.

Далі виберемо користувачів, які мають щонайменше по H товарів з кожного класу. З їхніх товарів виберемо по L товарів з класу “товар сподобався”, які мають найбільші значення r_{ui}, та по L товарів з класу “товар не сподобався”, які мають найменші значення r_{ui}. Це зроблено для того, щоб в тренувальній вибірці була однакова кількість представників з кожного класу і щоб модель краще могла вивчити випадки, коли товар найбільше сподобався, а коли найменше. Таким чином, ми обрали

множину пар користувач-товар $X_{U_H I_L}$, які будемо використовувати для навчання моделі 2 рівня.

Модель першого рівня для навчання використовувала лише факти взаємодії користувача з тим чи іншим товаром та мірою впевненості у вподобанні. Модель другого рівня може вивчити більш складні зв'язки, а також використовувати додаткову інформацію.

Для кожного товару i розглянемо вектор m_i, який містить додаткову інформацію про товар. Тоді модель другого рівня для набору ознак $[x_u, y_i, m_i]$ буде визначати мітку класу "товар сподобався" чи "товар не сподобався". Модель навчається на попередньо відібраних даних, сформованих для множини пар користувач-товар $X_{U_H I_L}$.

За допомогою навченої моделі 2 рівня обчислюємо ймовірності віднесення до класу "товар сподобається" для кандидатів, які ми отримали від моделі 1 рівня. Далі для кожного користувача u відбираємо K_2 товарів, які мають найбільшу ймовірність. Результатом роботи дворівневої моделі буде упорядкований перелік відібраних товарів, які ймовірно сподобаються користувачу. Схему роботи алгоритму можна побачити на рис. 2.3.

Рисунок 2.3 Схема відбору товарів для надання рекомендацій за допомогою розробленої дворівневої моделі

2.4 Висновки до розділу 2

В даному розділі було розглянуто математичні основи існуючих методів рекомендацій товарів на основі контентної та колаборативної фільтрації.

Також було проаналізовано методи оцінювання якості наданих рекомендацій, а саме методи оцінювання прогнозу оцінки товару, методи оцінювання якості рекомендацій.

Було запропоновано власний алгоритм запропоновано алгоритм дворівневої моделі для здійснення рекомендацій, який поєднує собі матричну факторизацію та градієнтний бустинг дерев. За допомогою матричної факторизації ми відбираємо кандидатів для рекомендації, за допомогою градієнтного бустингу дерев ми відбираємо остаточний перелік товарів, які ми будемо рекомендувати. Такий підхід дозволяє здійснювати релевантні рекомендації за прийнятний час.

РОЗДІЛ 3 АРХІТЕКТУРА РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ

3.1 Огляд використаних даних

Для розробки рекомендаційної системи було використано дані української компанії Megogo, яка є онлайн сервісом для перегляду фільмів. Компанія організовувала змагання Megogo Kaggle Challenge [21], в якому виклала у відкритий доступ анонімізовані дані своїх користувачів.

Тренувальний датасет складається з інформації про сесії переглядів фільмів та серіалів користувачами Megogo в Україні. Датасет містить інформацію про активність користувачів за липень, серпень та вересень 2018 року. Завданням для учасників змагання було передбачити контент, який користувачі подивляться в жовтні 2018 року. За перегляд вважається сеанс, у якого відсоток перегляду фільму більший за половину. Розподіл величини відсотку в навчальній вибірці можна побачити на рис. 3.1

Рисунок 3.1 - Розподіл переглянутих та не переглянутих відео

Тренувальний набір даних містить інформацію про 7964397 сесій переглядів, у якому 402009 унікальних користувачів та 7221 фільмів та серіалів.

Як бачимо на рис. 3.2, що 20% фільмів мають 80% переглядів, тобто користувачі дивляться найпопулярніші фільми, але є дуже багато інших, які могли б теж їм сподобатись, але вони про них не знають.

Рисунок. 3.2 - Розподіл кількості переглядів фільмів

Тестовий датасет містить в собі інформацію про 88% учасників з відомою історією в тренувальному наборі даних та 12% без історії (холодний старт).

3.2 Опис експерименту

Метою даної роботи було продемонструвати роботу дворівневої рекомендаційної моделі, схему якої зображено на рис. 3.3.

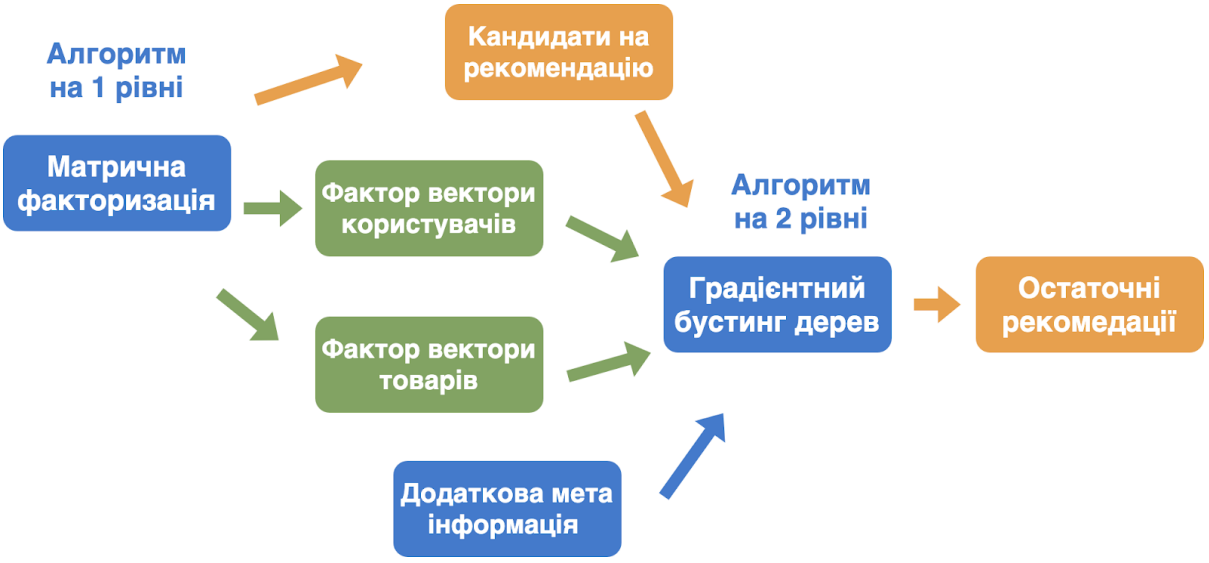


Рисунок 3.3 - Схема дворівневої моделі

На першому рівні було використано модель Implicit ALS. Спочатку з тренувальних даних була відібрана інформація про перегляди користувачами відео, яка складалася з ідентифікатора користувача, ідентифікатора відео та відсотку перегляду відео. За міру впевненості вподобання контенту було взято відсоток перегляду відео.

Приклад тренувальних даних зображено на рис. 3.4.

	user_id	video_id	watching_percentage
0	21603820	9583642	0.0839
1	35636970	24645936	0.5990
2	78312976	25397362	1.0000
3	122261599	5205267	0.9100
4	53477088	14098190	0.4850

Рисунок 3.4 - Приклад даних, які використовуються для тренування моделі 1 рівня

Результатом моделі став набір прихованих фактор векторів, які описували уподобання користувачів та характеристики товарів, які наведено на рис. 3.5 та рис. 3.6

	user_id	uf0	uf1	uf2	uf3	uf4	uf5	uf6	uf7	uf8	...	uf90
0	21603820	-0.071293	0.559415	-0.072539	0.129628	-0.280565	0.059878	-0.283093	-0.060836	0.450761	...	0.073832
1	35636970	0.078816	-0.080879	0.154771	-0.018486	1.056839	0.269800	0.520433	0.593316	0.011278	...	-0.358628
2	78312976	-0.249220	-0.237675	0.078369	0.563539	-0.011536	0.416894	-0.021369	-0.241337	0.021629	...	0.130717

Рисунок 3.5 - Вектор латентних факторів користувачів

	primary_video_id	if0	if1	if2	if3	if4	if5	if6	if7	if8	...	if90
0	9583642	-0.099971	-0.049025	0.043438	-0.019529	-0.075116	0.004447	-0.097199	0.012962	0.171468	...	0.197261
1	24645936	0.019838	0.006003	-0.007013	-0.035202	0.112414	-0.047919	0.020665	0.049804	-0.024089	...	-0.022884
2	25397362	-0.097113	-0.008341	0.059431	0.133416	-0.030762	0.040883	0.019127	-0.072346	-0.015360	...	0.084288

Рисунок 3.6 - Вектор латентних факторів відео

Також на 1 рівні було отримано набір кандидатів для рекомендації, які мають найбільшу ймовірність сподобатись. Приклад рекомендацій для користувача з ідентифікатором 235599 зображено на рис. 3.7.

	user_id	video_id	score
0	235599	21757334	0.929609
1	235599	6597685	0.912704
2	235599	33501500	0.878318
3	235599	4889342	0.824930
4	235599	22328483	0.800370
5	235599	14540399	0.799846
6	235599	3636342	0.783520
7	235599	10979465	0.771781
8	235599	13536958	0.769564
9	235599	15117563	0.750291

Рисунок 3.7 - Приклад відео які були зарекомендовані моделлю на 1 рівні

Для навчання моделі на 2 рівні спочатку було підготовлено навчальну вибірку. Тренувальні дані було поділено на 2 класи: “відео сподобалось” та “відео не сподобалось”. Далі було відібрано користувачів, які мали взаємодію з щонайменше по 10 відео з кожного класу. З їхніх відео було вибрано по 5 відео з класу “відео сподобалось”, які мали найбільші значення відсотка перегляду відео, та по 5 відео з класу “відео не сподобалось”, які мали найменший відсоток. Ці дії було зроблено для того, щоб в навчальній вибірці отримати однаковий розподіл класів, зменшити розмір навчальних даних, відібрати яскраво виражені позитивні та негативні випадки.

Додаткова дані про відео містили інформацію про назву відео, тип відео, рік, країна, якість, вікове обмеження, жанр та оцінки з різних ресурсів. Приклад додаткових даних зображено на рис. 3.8

	primary_video_id	title	type	year	country	rating_imdb	rating_kinopoisk	score_by_popular	score_by_recommended	quality	age_limit	for_kids
0	20069583	Мой лучший друг	FILM	2006	Франция	6.7	7.417	0.0	9283.0	FullHD	16	0
1	11492142	Бэндслэм	FILM	2009	США	6.3	6.766	16.0	8238.0	FullHD	12	0
2	24885597	Толстяки	FILM	2009	Испания	6.7	6.962	9.0	6700.0	HD	16	0

Рисунок 3.8 - Приклад додаткових даних про відео

Отже на першому рівні було отримано фактор вектори, які описують уподобання користувачів та характеризують відео. Ці вектори було застосовано під час навчання та обрахунку рекомендацій для моделі 2 рівня, як зображено на рис. 3.9.

Рисунок 3.9 - Схема використання даних з першого рівня на другому рівні

Для невідомих користувачів було присвоєно усереднений фактор вектор по всім відомим користувачам. Було проведено експеримент, під час якого було скалярно перемножено усереднений вектор користувачів на фактор вектори всіх фільмів, отримано значення міри вподобання для кожного фільму. Фільми які мали найбільше значення співпадали з фільмами, які мали найбільшу популярність. Це вирішувало проблему “холодного старту”.

3.3 Аналіз отриманих результатів

Порівняння результатів якості рекомендацій розробленої дворівневої моделі в порівнянні з рекомендацією, яка містить лише найпопулярніші товари. Результати метрик оцінювання наведені в таблиці 3.1 та зображені на рис. 3.10.

Таблиця 3.1 - Результати оцінювання моделей

	Precision@10	Recall@10	MAP@10	NDCG@10
Модель 2 рівня	0.0425413	0.113818	0.0718531	0.0945739
Модель 1 рівня	0.038066	0.106846	0.0541142	0.0861713
Найпопуляр ніші товари	0.0130161	0.023471	0.00832245	0.0191037

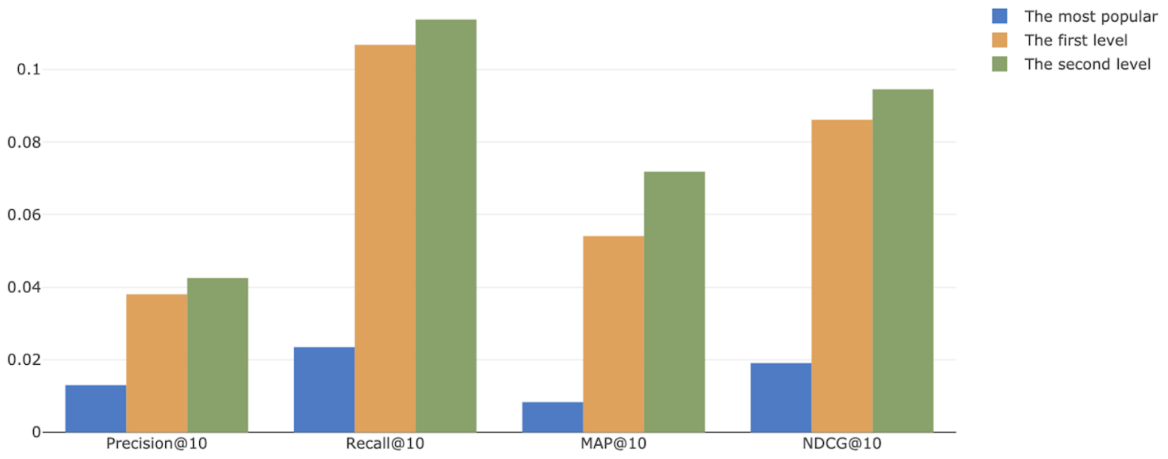


Рисунок 3.10 - Значення метрик якості рекомендацій

3.4 Демонстрація розробленого програмного продукту

Для демонстрації роботи дворівневої моделі рекомендацій було створено програмний продукт за допомогою мови програмування Python

та за використання фреймворку flask. Схему структури програмного продукту наведено на рис. 3.11.

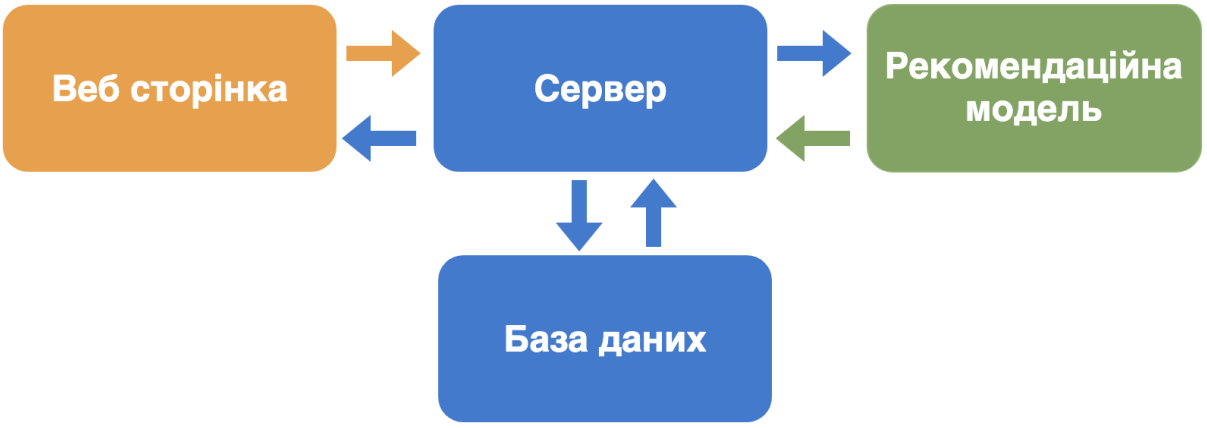


Рисунок 3.11 - Схема структури програмного продукту

Спочатку було розроблено систему веб сторінок для наочної демонстрації рекомендацій та взаємодії з користувачем. Було розроблено сервер, який взаємодіє з веб сторінками, базою даних та з рекомендаційною моделлю. З веб сторінки сервер отримує взаємодії користувача з товарами, які він записує в базу даних. З бази даних сервер передає в рекомендаційну модель всю необхідну інформацію, яка необхідна для здійснення рекомендації. Рекомендаційна модель повертає серверу ідентифікатори рекомендованих товарів. З бази даних сервер отримує необхідну інформацію про зарекомендовані товари, яку він потім відображає на веб сторінках. З веб сторінки ми можемо отримати відгук які з рекомендованих товарів сподобались користувачу, записати інформацію в базу даних та підрахувати метрики.

Вигляд веб сторінок розробленого програмного продукту зображено на рис. 3.12 - 3.22

Рисунок 3.12 - Початкова сторінка програмного продукту

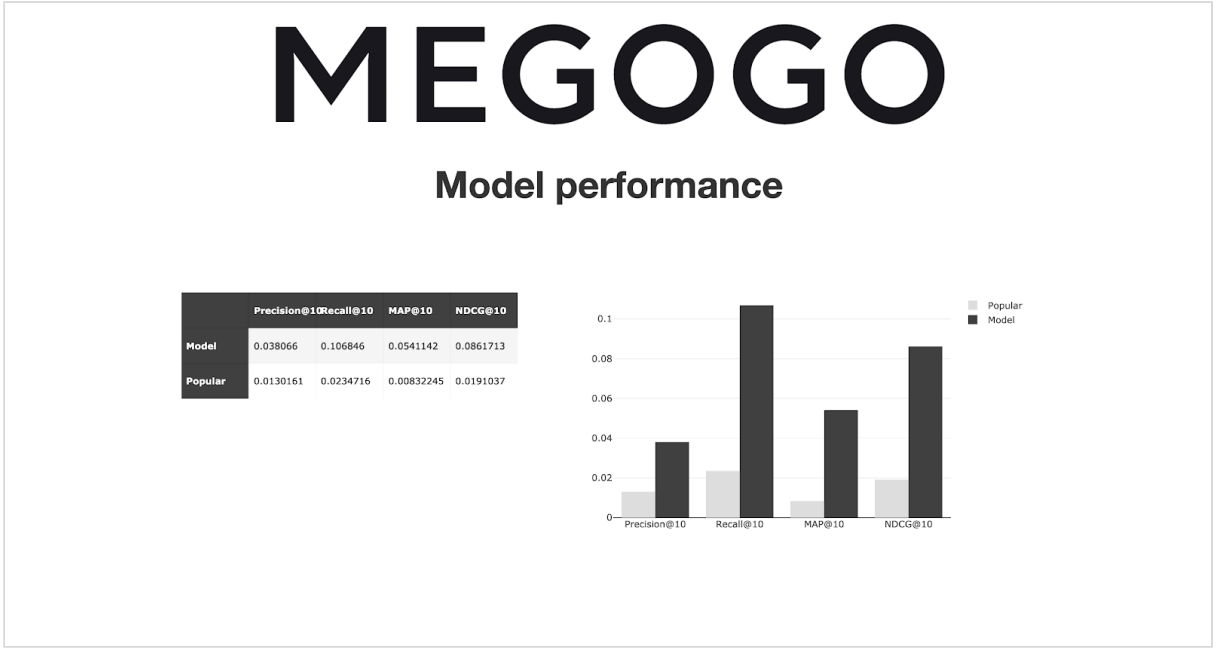
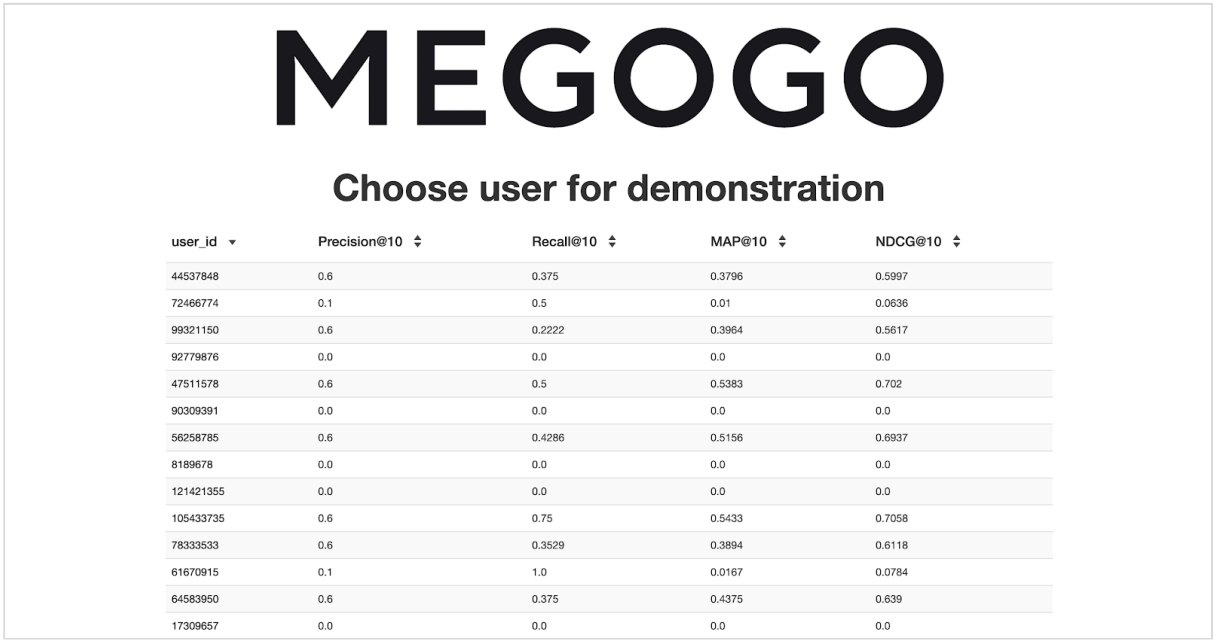


Рисунок 3.13 - Порівняння результатів роботи моделі



user_id	Precision@10	Recall@10	MAP@10	NDCG@10
44537848	0.6	0.375	0.3796	0.5997
72466774	0.1	0.5	0.01	0.0636
99321150	0.6	0.2222	0.3964	0.5617
92779876	0.0	0.0	0.0	0.0
47511578	0.6	0.5	0.5383	0.702
90309391	0.0	0.0	0.0	0.0
56258785	0.6	0.4286	0.5156	0.6937
8189678	0.0	0.0	0.0	0.0
121421355	0.0	0.0	0.0	0.0
105433735	0.6	0.75	0.5433	0.7058
78333533	0.6	0.3529	0.3894	0.6118
61670915	0.1	1.0	0.0167	0.0784
64583950	0.6	0.375	0.4375	0.639
17309657	0.0	0.0	0.0	0.0

Рисунок 3.14 - Метрики якості рекомендацій для кожного користувача системи

На рис. 3.15 наведено приклад демонстрації роботи алгоритму для певного користувача компанії Megogo. На сторінці зображені відео, які були в навчальній вибірці користувача, фільми які були в тестовій вибірці, а саме ті, які він подивився в жовтні 2018 року, та наведено демонстрацію рекомендованих товарів за допомогою дворівневої моделі. В чорній рамці наведено ті фільми, які модель зарекомендувала і користувач дійсно подивився їх, тобто ті які знаходиться і в тестовій вибірці.

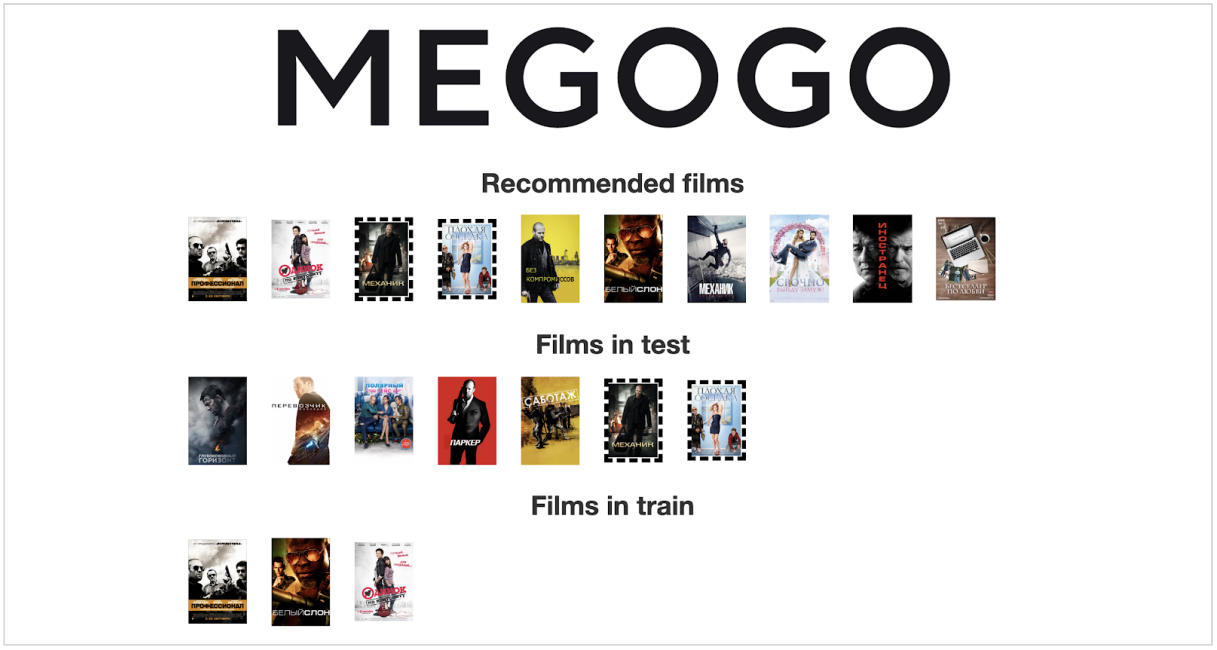


Рисунок 3.15 - Демонстрації роботи алгоритму для певного користувача

Рисунок 3.16 - Приклад сторінки відео

Рисунок 3.17 - Сторінка вибору уподобань нового користувача

Рисунок 3.18 - Сторінка очікування тренування моделі

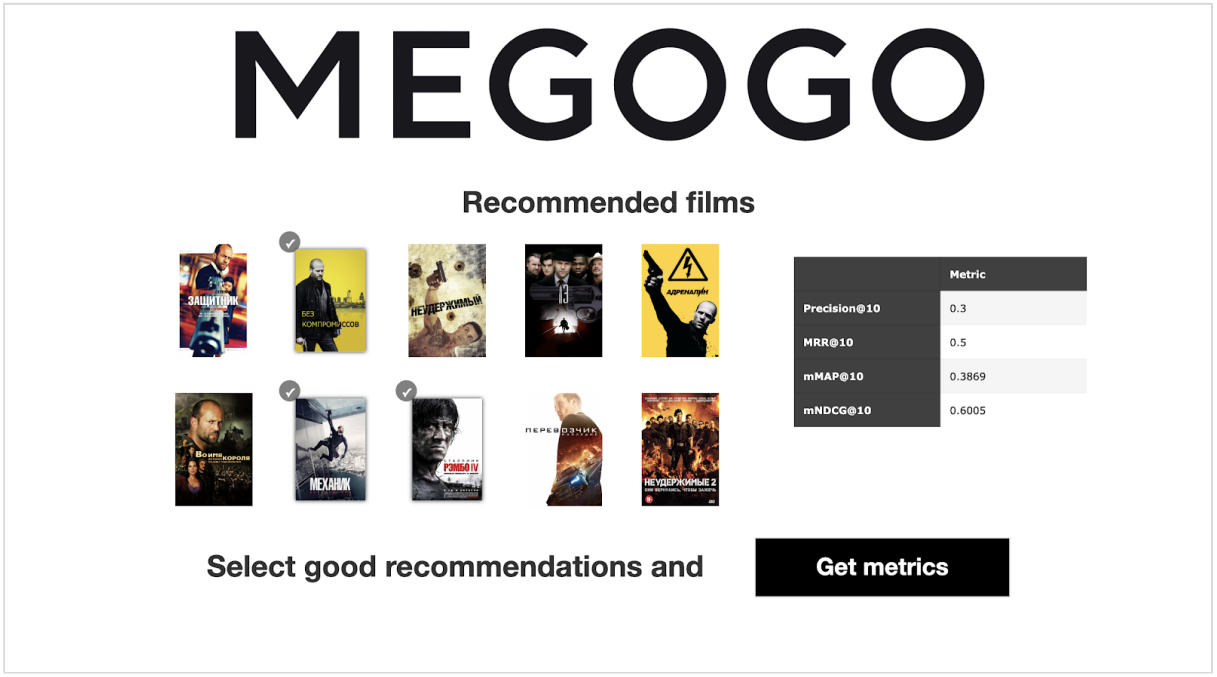


Рисунок 3.19 - Сторінка демонстрації рекомендацій для нового користувача

3.5 Висновки до розділу 3

Було розроблено програмний продукт за допомогою мови програмування python та фреймворку flask. Для побудови рекомендацій було використано такі бібліотеки машинного навчання, як numpy, pandas, sklearn, implicit та CatBoost. Розроблений програмний продукт дає змогу на основі вибраного контенту, який сподобався користувачу, отримати список рекомендацій фільмів та відео.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

Створення програмного продукту (ПП) зазвичай потребує значних трудових та фінансових витрат. Щоб ефективно використовувати наявні ресурси, спочатку необхідно провести аналіз можливих варіантів створення ПП, який дозволить обрати серед них найбільш раціональний.

Функціонально-вартісний аналіз — це метод комплексного техніко-економічного дослідження об’єкта з метою розвитку його корисних функцій при оптимальному співвідношенні між їхньою значимістю для споживача і витратами на їхнє здійснення. Є одним з основних методів оцінки вартості науково-дослідної роботи, оскільки ФВА враховує як технічну оцінку продукту, що розробляється, так і економічну частину розробки. Крім того, даний метод дозволяє вибрати оптимальний, як з погляду розробника, так і з точки зору покупця варіант розв’язання будь-якої задачі, а також дозволяє оптимізувати витрати й час виконання робіт. Зниження витрат виробництва треба починати з аналізу властивостей виробу, що використовуються, а також технічних функцій його складових частин.

У даній роботі проводиться оцінювання основних характеристик програмного продукту, який здійснює рекомендацію товарів для споживачів та наочно візуалізує результати.

Врахувавши економічні фактори та характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням, нижче наведено аналіз різних варіантів реалізації програмного модулю з метою вибору найбільш оптимального варіанту.

ФВА складається з декількох етапів:

- Спочатку визначається послідовність функцій, необхідних для виробництва продукту. Далі вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. Також на даному етапі оптимізується послідовність завдяки скороченню кроків, що не впливають на цінність та витрати продукту.
- На наступному етапі для кожної функції визначаються повні річні витрати та кількість робочих годин.
- Далі на основі оцінок попереднього пункту для кожної функції визначається кількісна характеристика джерел витрат;
- Наостанок, проводиться остаточний розрахунок витрат для створення продукту.

4.1. Обґрунтування функцій програмного продукту

Головна функція F0 – це розробка програмного продукту, який дозволяє рекомендувати товари користувачам. Виходячи з конкретної мети, можна виділити наступні основні функції програмного продукту:

F1 – вибір мови програмування

F2 – вибір алгоритму рекомендації

F3 – вибір бібліотеки для візуалізації

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування Python;

б) мова програмування C++;

Функція F_2:

а) формат .csv;

б) формат .json;

Функція F_3:

а) алгоритм матричної факторизації;

б) алгоритм *K* найближчих сусідів.

Варіанти реалізації основних функцій наведені у морфологічній карті системи зображеній на рис 4.1.

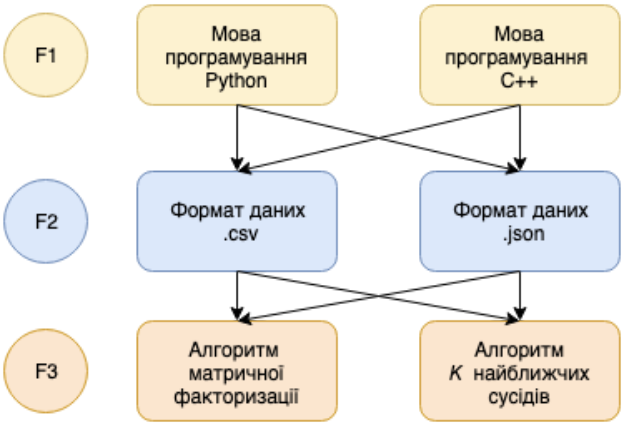


Рисунок 4.1 Морфологічна карта

Морфологічна карта відображає всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП. На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій, які наведені в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	*A*	Менший об’єм програмного коду, багато спеціалізованих	Помірна швидкість виконання коду

		бібліотек для обробки даних, легкість у написанні коду	
	Б	Код швидко виконується	Великий обсяг написання коду, більше часу на розробку продукту, мала кількість реалізованих бібліотек для обробки даних
F2	*А*	розповсюджений формат, зручний у представленні табличних даних	Складність передавати через АПІ продукту
	Б	зручно передавати через АПІ	менш зручний для представлення даних у табличному форматі
F3	*А*	Висока точність та швидкість, можливість обробки великого обсягу даних, можливість працювати з явними та неявними відгуками	Складність в реалізації
	Б	Легкість у розумінні	Повільно рекомендує, можливість працювати лише з явними відгуками

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F1:

Оскільки час та обсяг написання програмного коду, наявність спеціалізованих бібліотек з машинного навчання для обробки великої кількості даних є більш вагомим фактором, ніж лише швидкість виконання, то варіант б) можна не розглядати.

Функція F2:

Так, як наведені варіанти реалізації мають свої значні переваги один над одним, варто розглянути та проаналізувати обидва.

Функція F3:

Оскільки нам важливо вміти працювати з явними та неявними оцінками, точність та швидкість виконання, то варіант б) можна не розглядати.

Таким чином будемо використовувати наступні варіанти реалізації програмного продукту:

1) F1а–F2а–F3а

2) F1а–F2б–F3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2. Обґрунтування системи параметрів програмного продукту

4.2.1. Опис параметрів

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

1) *X1* – об’єм пам’яті для збереження даних;
2) *X2* – час обробки даних;
3) *X3* – точність розв’язку;
4) *X4* – потенційний об’єм програмного коду.

X1 – відображає об’єм пам’яті в оперативній пам’яті ПК, необхідний для збереження та обробки даних під час виконання програми.

X2 – відображає час, який витрачається на дії.

X3 – відображає точність розв’язку для метрики ранжування

X4 – показує розмір програмного коду який необхідно створити безпосередньо розробнику.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 2.

Таблиця 4.2 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Об’єм пам’яті для збереження даних	X1	Мб	32	16	8
Час обробки даних алгоритмом	X2	мс	800	420	60
Точність розв’язку	X3	доля одиниці	10E-1	10E-2	10E-3
Потенційний об’єм	X4	кількість	2000	1500	1000

програмного коду		рядків коду			

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

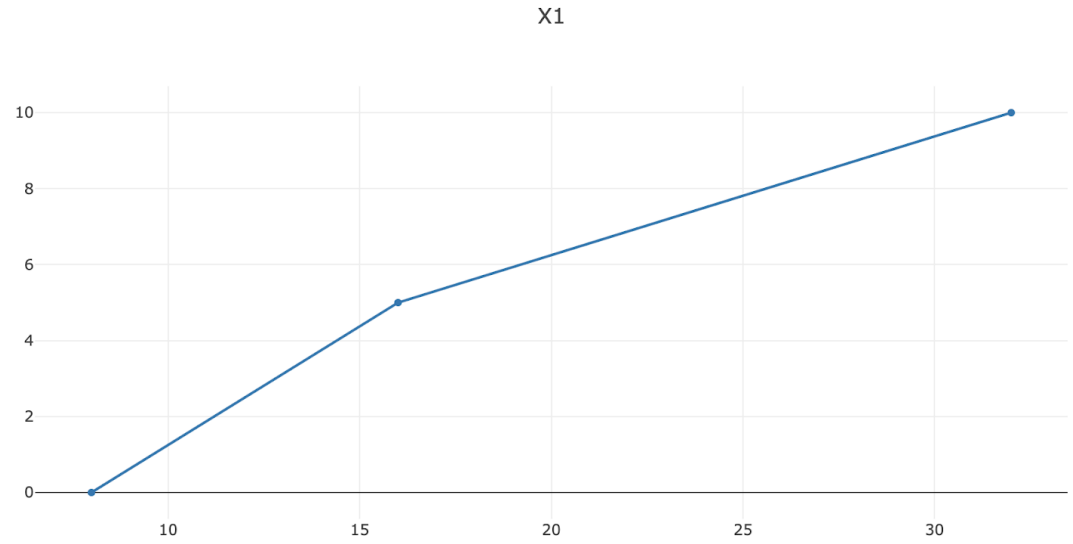


Рисунок 4.2 – X1, об’єм пам’яті для збереження даних

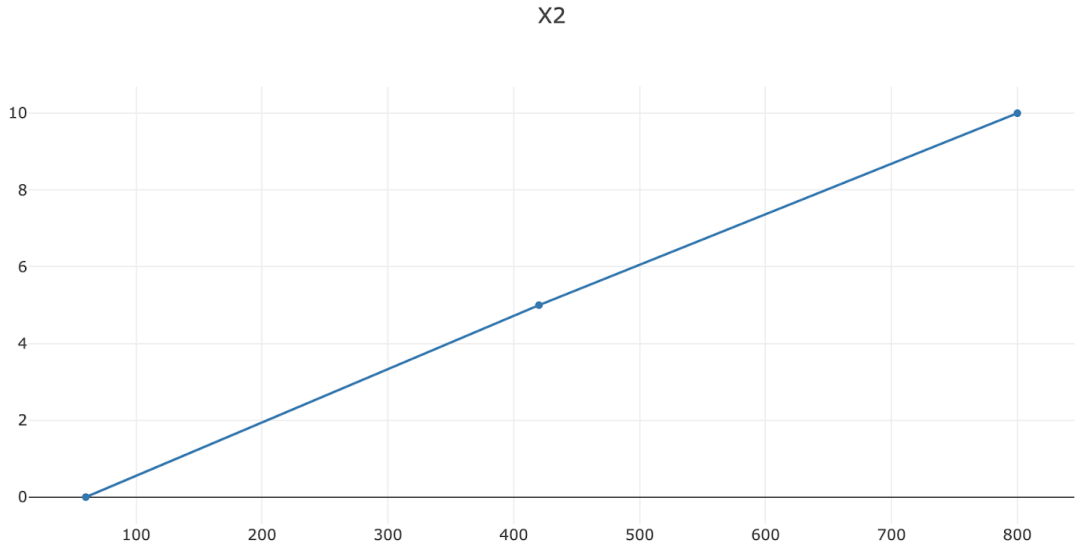


Рисунок 4.3 – X2, час обробки даних алгоритмом

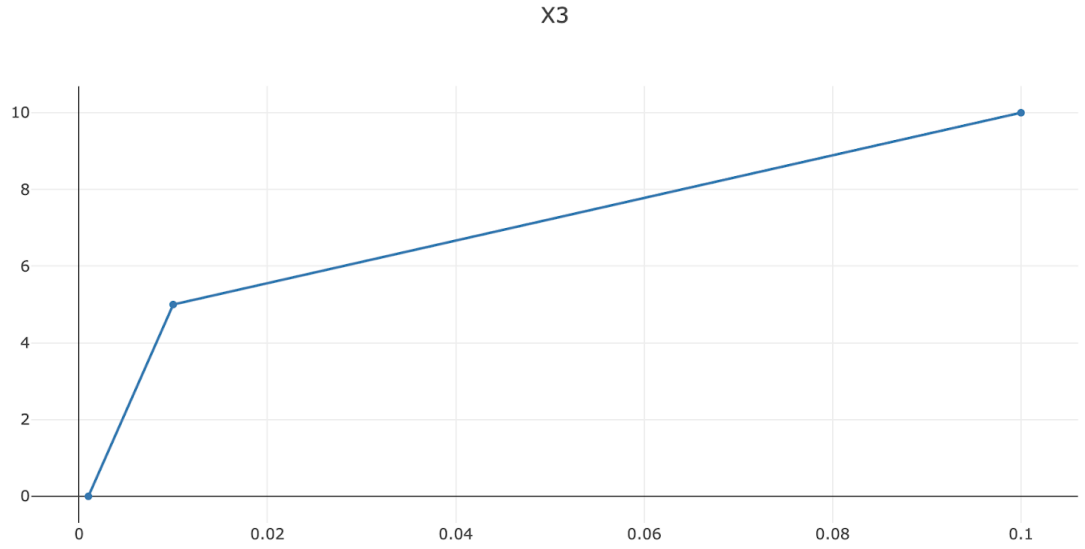


Рисунок 4.4 – X3, точність розв’язку

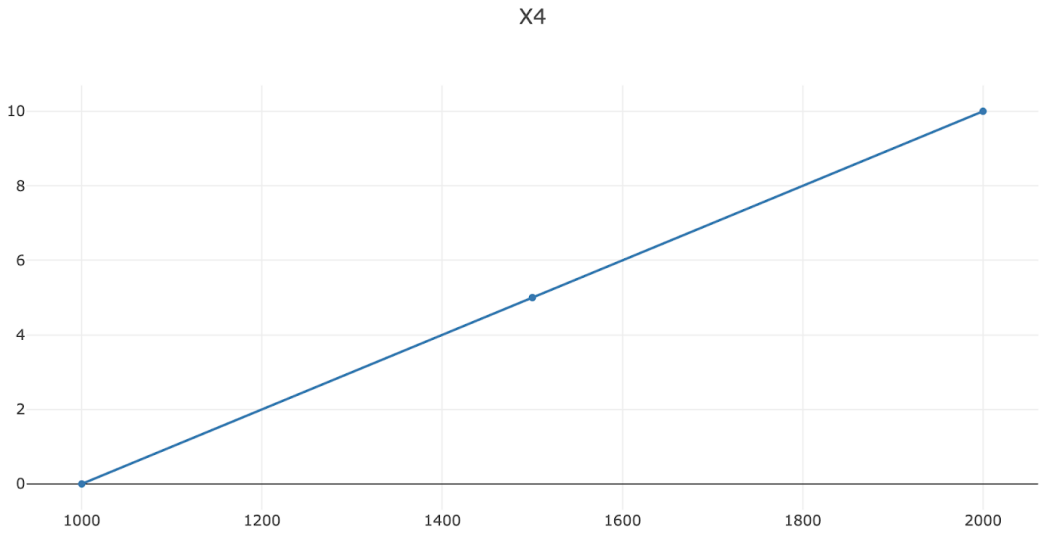


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який найбільш точно надає рекомендації

товарів для користувачів.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значущості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
	1	2	3	4	5	6	7			
X1	3	4	2	3	4	3	4	21	5.5	30.25
X2	1	2	1	1	2	2	2	12	-6.5	42.25
X3	2	1	3	2	1	1	1	11	-6.5	42.25
X4	4	3	4	4	3	4	3	26	7.5	56.25
	10	10	10	10	10	10	10	70	0	171

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70$$

де N – число експертів, n – кількість параметрів.

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 171$$

Обчислимо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 171}{7^2(4^3 - 4)} = 0.69 > W_k = 0.67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.

Таблиця 4.4 - Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	>	>	>	>	>	1.5
X1 і X3	>	>	<	>	>	>	>	>	1.5
X1 і X4	<	>	<	<	>	<	>	<	0.5
X2 і X3	<	>	<	<	>	>	>	>	1.5
X2 і X4	<	<	<	<	<	<	<	<	0.5
X3 і X4	<	<	<	<	<	<	<	<	0.5

Числове значення, що визначає ступінь переваги *і*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j. \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за формулою (4.1):

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \qquad (4.1)$$

$$b_i = \sum_{j=1}^{n} a_{ij}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступною формулою (4.2):

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'} \qquad (4.2)$$

$$b_i' = \sum_{j=1}^{n} a_{ij} b_j \, .$$

Як видно з таблиці 5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i x_i	Параметри x_j x_j				Перша ітерація		Друга ітерація		Третя ітерація	
	1	2	3	4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	1,5	1,5	0,5	4.5	0.281	16.25	0.275	59.125	0.274
X2	0,5	1,0	1,5	0,5	3.5	0.189	12.25	0.208	44.875	0.208
X3	0,5	0,5	1,0	0,5	2.5	0.156	9.25	0.157	34.125	0.158
X4	1,5	1,5	1,5	1,0	5.5	0.344	21.25	0.360	77.875	0.360
Всього:					16	1	59	1	216	1

4.4 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо. Абсолютні значення параметрів *X1*(об’єм пам’яті для збереження даних) відповідають технічним вимогам умов функціонування даного ПП. Абсолютне значення параметра *X2 (*час обробки даних) обрано

не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 800 мс або варіанту б) 80мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi} B_{ij}$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X4	1100	1	0.344	0.344
F2	А	X1	12	4	0.281	1.124
		X2	420	5	0.189	0.945
	Б	X1	15	5	0.281	1.405
		X2	350	4	0.189	0.756
F3	А	X3	10E-2	5	0.156	0.78

За даними з таблиці 4.6 та за формулою (4.3) визначаємо рівень якості кожного з варіантів:

$$K_K = K_{TY}[F_{ik}] + K_{TY}[F_{2k}] + ... + K_{TY}[F_{zk}] \quad (4.3)$$

$$K_{K1} = 1.124+0.945+0.78+ 0,344 = 3.193$$

$$K_{K2} = 1.405+0.756+0.78+ 0,344 = 3,285$$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.5 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1) Розробка проекту програмного продукту;
2) Розробка користувацького інтерфейсу для демонстрації;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється так:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.4)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_{М}$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеня новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_{Р}$ =90 людино-днів.

Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_{П}$ = 1.7.

Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1:

$К_{СК}$ = 1.

Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта:

$К_{СТ}$ = 0.7.

Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_{1} = 90 \cdot 1.7 \cdot 0.7 = 107.1 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших досліджень за формулою (4.4). Для другого завдання (використовується алгоритм третьої групи складності, ступінь новизни Б), тобто $Т_{Р}$ =27 людино-днів, $К_{П}$ =0.8, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$Т_{2} = 27 \cdot 0.8 \cdot 0.8 = 17.28 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$T_I = (107.1 + 17.28 + 4.8 + 17.28) \cdot 8 = 1171{,}68$ людино-годин;

$T_{II} = (107.1 + 17.28 + 6.91 + 17.28) \cdot 8 = 1188.56$ людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь інженер даних та веб розробник з окладом 6000 грн. кожен, а також один інженер з машинного навчання з окладом 12000 грн.

Визначимо зарплату за годину за формулою:

$$C_Ч = \frac{М}{T_m \cdot t} \text{ грн.} \quad (4.5)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів в тиждень;

t – кількість робочих годин в день.

Обчислимо зарплату за годину за формулою (4.5):

$$C_Ч = \frac{6000 + 6000 + 12000}{3 \cdot 21 \cdot 8} = 47.61 \text{ грн}$$

Визначимо заробітну плату за формулою

$$C_{ЗП} = C_Ч \cdot T_i \cdot K_Д\,, \quad (4.6)$$

де $C_ч$– величина погодинної оплати праці для робітника;

T_i – трудомісткість відповідного завдання;

$К_д$ – норматив, який враховує додаткову заробітну плату.

Зарплату розробників за варіантами обчислимо за формулою (4.6):

I. $С_{ЗП} = 47.61 \cdot 1171.68 \cdot 1.2 = 66953.14$ грн.

II. $С_{ЗП} = 47.61 \cdot 1188.56 \cdot 1.2 = 67904.80$ грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 66953.14 \cdot 0.22 = 14729$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 67904.80 \cdot 0.22 = 14939$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного розробника з окладом 6000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 6000 \cdot 0,2 = 14400 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 14400 \cdot (1 + 0.2) = 17280 \text{ грн.}$$

Відрахування на єдиний соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 17280 \cdot 0.22 = 3801.6 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20 000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 20\,000 = 5750 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 20\,000 \cdot 0.05 = 1150 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot К_В = (365 - 104 - 8 - 16) \cdot 8 \cdot 0.9 = 1706.4 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706.4 \cdot 0.156 \cdot 0.78 \cdot 2.7515 = 571.30 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 20000 \cdot 0,67 = 13400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$$С_{ЕКС} = 17280 + 3801.6 + 5750 + 1150 + 571.30 + 13400 = 41952.9 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 41952.9/1706,4 = 24.58 \text{ грн/час.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М\text{-}Г} \cdot Т$$

I. $С_М = 24.58 \cdot 1171,68 = 28806.4$ грн.

II. $С_М = 24.58 \cdot 1188.56 = 29214.8$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

I. $С_Н = 66953.14 \cdot 0,67 = 44858.60$ грн.

II. $С_Н = 67904.80 \cdot 0,67 = 45396.22$ грн.

Отже, вартість розробки ПП становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$$

I. $С_{ПП} = 66953.14 + 24618 + 28806.4 + 44858.60 = 165236.14$ грн.

II. $С_{ПП} = 67904.80 + 24968 + 29214.8 + 45396.22 = 167483.82$ грн.

4.6. Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$$К_{ТЕР1} = 3.193 / 165236.14 = 1.932 \cdot 10^{-5},$$

$$К_{ТЕР2} = 3,285 / 167483.82 = 1.961 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 1.961 \cdot 10^{-5}$.

4.7 Висновки до розділу 4

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломної роботи. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації. На основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП. Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко економічного рівня якості КТЕР = $1.941 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Мова програмування – Python;
- завантаження та збереження даних у файли формату *.json;
- використовуємо алгоритм матричної факторизації

Даний варіант виконання програмного продукту дає користувачу досить точну швидку рекомендаційну систему.

ВИСНОВКИ

В даній дипломній роботі розв’язано задачу побудови рекомендаційної системи товарів на основі моделі вподобань користувачів.

В роботі отримані наступні результати:

- Проведено порівняння існуючих підходів до формування списку рекомендацій, проаналізовано їх переваги і недоліки;
- Реалізовано алгоритм колаборативної фільтрації на основі пошуку прихованих факторів;
- Реалізовано дворівневу модель рекомендацій товарів для користувачів. Проаналізовано якість роботи розробленого алгоритму;
- Практичним результатом роботи є розроблена рекомендаційної системи для онлайн кінотеатру Megogo на основі запропонованого алгоритму у вигляді прикладної програми. Розроблена система формує список рекомендацій фільмів для існуючих користувачів сервісу.

СПИСОК ЛІТЕРАТУРИ

1. Recommendation Systems – How Companies are Making Money URL: https://sigmoidal.io/recommender-systems-recommendation-engine/ (Last accessed: 29.05.2019).
2. Automatic movie ratings prediction using machine learning URL: http://www.csc.kth.se/~miksa/papers/AutomaticMovieRatingsPrediction_MIPRO.pdf (Last accessed: 30.05.2019).
3. The ACM Conference Series on Recommender Systems URL: https://recsys.acm.org/ (Last accessed: 27.05.2019).
4. F. Ricci, L. Rokach, S. Bracha. Recommender Systems Handbook. New York: Springer, 2015. 1003 p.
5. Statistical Analysis of K-Nearest Neighbor Collaborative Recommendation URL: https://arxiv.org/pdf/1010.0499.pdf (Last accessed: 29.05.2019).
6. Montaner, M., López, B., de la Rosa, J.L.: A taxonomy of recommender agents on the internet. Artificial Intelligence Review 19(4), 285–330 (2003) (Last accessed: 29.05.2019).
7. Artificial Intelligence in Retail – 10 Present and Future Use Cases URL: https://emerj.com/ai-sector-overviews/artificial-intelligence-retail/ (Last accessed: 30.05.2019).
8. Query Dependent Ranking Using K-Nearest Neighbor URL: https://www.andrewoarnold.com/fp025-geng.pdf (Last accessed: 29.05.2019).
9. Снитюк В. Є. Прогнозирование. Модели, методы, алгоритмы: учебное пособие. Київ: Маклаут, 2008. 364 с.
10. Similarity Measures used in Recommender Systems: A Study URL:

https://pdfs.semanticscholar.org/943a/e455fafc3d36ae4ce68f1a60ae4f85623e2a.pdf (Last accessed: 06.06.2019).

11. O'Connor M. PolyLens: A Recommender System for Groups of Users URL: http://files.grouplens.org/papers/poly-camera-final.pdf (Last accessed: 25.05.2019).
12. Matrix Factorization Techniques for recommender systems URL: https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf (Last accessed: 10.05.2019).
13. A Singularly Valuable Decomposition: The SVD of a Matrix URL: https://datajobs.com/data-science-repo/SVD-[Dan-Kalman].pdf (Last accessed: 04.06.2019).
14. Large-Scale Machine Learning with Stochastic Gradient Descent URL: https://leon.bottou.org/publications/pdf/compstat-2010.pdf (Last accessed: 08.06.2019).
15. Collaborative Filtering for Implicit Feedback Datasets URL: http://yifanhu.net/PUB/cf.pdf (Last accessed: 01.06.2019).
16. Онлайн курс "Навчання на розмічених даних" URL: https://www.coursera.org/learn/supervised-learning?specialization=machine-learning-data-analysis (дата звернення: 25.05.2019).
17. A Survey on Decision Tree Algorithms of Classification in Data Mining URL:https://www.researchgate.net/publication/324941161_A_Survey_on_Decision_Tree_Algorithms_of_Classification_in_Data_Mining (Last accessed: 30.05.2019).
18. XGBoost: A Scalable Tree Boosting System URL: https://arxiv.org/pdf/1603.02754.pdf (Last accessed: 28.05.2019).

19. Метрики качества ранжирования [Електронний ресурс] // HABR. – 2016 URL: https://m.habr.com/ru/company/econtenta/blog/303458/. (Last accessed: 28.05.2019).
20. CatBoost: unbiased boosting with categorical features URL: https://arxiv.org/pdf/1706.09516.pdf (Last accessed: 30.05.2019).
21. Megogo Kaggle Challenge URL: https://www.kaggle.com/c/megogochallenge/overview (Last accessed: 05.06.2019).

ДОДАТОК А Код програмного продукту

```
import pandas as pd
import numpy as np
import copy

class Metrics():
    def __init__(self, test, predict, calculate_rating_metrics = False, verbose = False):

        users_in_test = set()
        users_in_predict = set()
        self.test_item = {}
        self.predict_item = {}
        self.type_predict = ''

        if type(test) == type(pd.DataFrame()):
            test[['user', 'item']] = test[['user', 'item']].astype(str)
            users_in_test = set(test.user.unique())
            self.test_item = test.groupby(['user'])['item'].apply(lambda grp: list(grp)).to_dict()
            if verbose:
                print('type of test: pandas')

        if type(predict) == type(pd.DataFrame()):
            predict[['user', 'item']] = predict[['user', 'item']].astype(str)
            users_in_predict = set(predict.user.unique())
            self.predict_item = predict.groupby(['user'])['item'].apply(lambda grp: list(grp)).to_dict()
            self.type_predict = 'pandas'
            if verbose:
                print('type of predict: pandas')

        if type(test) == type({}):
            self.test_item = test
            users_in_test = set(list(test.keys()))
            if verbose:
                print('type of test: dict')

        if type(predict) == type({}):
            self.predict_item = predict
```

```
        users_in_predict = set(list(predict.keys()))
        self.type_predict = 'dict'
        if verbose:
            print('type of predict: dict')


    if type(list(self.test_item.keys())[0]) != type(list(self.predict_item.keys())[0]):
        print('ERROR: different type for users in dict')

    if type(list(self.test_item.values())[0][0]) != type(list(self.predict_item.values())[0][0]):
        print('ERROR: different type for items in dict')

    self.users = list(users_in_test.intersection(users_in_predict))
    if verbose:
        print('amount of users in test:', len(users_in_test))
        print('amount of users in predict:', len(users_in_predict))
        print('amount of common users:', len(self.users))


    self.metrics_for_recommend = ['Precision', 'Recall', 'mMAP', 'MAP', 'NDCG', 'mNDCG', 'MRR']
    self.metrics_for_rating = ['RMSE', 'MAE', 'MSE']

    self.available_metrics =  copy.copy(self.metrics_for_recommend)

    #for Normed Entropy
    if self.type_predict == 'pandas':
        self.available_metrics.append('Normed_entropy')
        self.proportions = (predict.item.value_counts() / predict.shape[0]).sort_values().values
        self.num_predicted_items = len(predict.item.unique())

    self.saved_result = {}

    # use rating metrics
    self.calculate_rating_metrics = calculate_rating_metrics
            if calculate_rating_metrics and type(test) == type(pd.DataFrame()) and type(predict) ==
type(pd.DataFrame()):
        test.rating = test.rating.astype(float)
        predict.rating = predict.rating.astype(float)
```

```
        self.available_metrics.extend(self.metrics_for_rating)
        self.test_rating = test.groupby(['user'])['rating'].apply(lambda grp: list(grp)).to_dict()
        self.predict_rating = predict.groupby(['user'])['rating'].apply(lambda grp: list(grp)).to_dict()
        self.common_rating_test = {}
        self.common_rating_predict = {}
        for user in self.users:
            test_item = self.test_item[user]
            predict_item = self.predict_item[user]
            common_items= list(set(test_item).intersection(set(predict_item)))

            mask = np.isin(test_item, common_items)
            self.common_rating_test[user] = np.array(self.test_rating[user])[mask]

            mask = np.isin(predict_item, common_items)
            self.common_rating_predict[user] = np.array(self.predict_rating[user])[mask]

    def Precision(self, k, metric_per_user=False):

        return self.calculate_metric_for_recommend(self.precision_for_user, k, metric_per_user)

    def Recall(self, k, metric_per_user=False):
        return self.calculate_metric_for_recommend(self.recall_for_user, k, metric_per_user)

    def MAP(self, k, metric_per_user=False):
        return self.calculate_metric_for_recommend(self.map_for_user, k, metric_per_user)

    def mMAP(self, k, metric_per_user=False):
            return self.calculate_metric_for_recommend(self.map_for_user, k, metric_per_user, modified_metric =
True)

    def NDCG(self, k, metric_per_user=False):
            return self.calculate_metric_for_recommend(self.ndcg_for_user, k, metric_per_user, modified_metric =
False)

    def mNDCG(self, k, metric_per_user=False):
            return self.calculate_metric_for_recommend(self.ndcg_for_user, k, metric_per_user, modified_metric =
True)
```

```
    def MRR(self, k, metric_per_user=False):
        return self.calculate_metric_for_recommend(self.mrr_for_user, k, metric_per_user)

    def calculate_metric_for_recommend(self, metric_function, k, metric_per_user=False, modified_metric =
False):
        user_metric = {}
        sum_metric_for_all_users = 0.

        for user in self.users:

            metric = 0.
            if modified_metric:
                metric = metric_function(user, k, modified_metric)
            else:
                metric = metric_function(user, k)

            user_metric[user] = metric
            sum_metric_for_all_users += metric

        result_metric = sum_metric_for_all_users/len(self.users)

        if metric_per_user:
            return result_metric, user_metric
        else:
            return result_metric

    def precision_for_user(self, user, k):

        return len(set(self.test_item[user]).intersection(set(self.predict_item[user])))/float(k)

    def recall_for_user(self, user, k):
        true_items = self.test_item[user]

        return len(set(true_items).intersection(set(self.predict_item[user])))/float(len(true_items))

    def map_for_user(self, user, k, modified_metric = False):
        true_items = self.test_item[user]
        predicted_items = self.predict_item[user]
```

```
    tru_predicted = 0.
    precisions = np.zeros(k)
    for i in range(k):
        if predicted_items[i] in true_items:
            tru_predicted += 1
            precisions[i] = tru_predicted/(i+1)

    average_precision = 0.
    if modified_metric:
        l = len(true_items)
        average_precision = sum(precisions)/min(k, l)

    else:
        average_precision = sum(precisions)/k

    return average_precision


def ndcg_for_user(self, user, k, modified_metric = False):
    true_items = self.test_item[user]
    predicted_items = self.predict_item[user]

    idcg = 0.

    l = k
    if modified_metric:
        l = min(len(true_items), k)

    for i in range(l):
        idcg += 1./np.log2(i+2)

    dcg = 0.
    for i in range(k):
        if predicted_items[i] in true_items:
            dcg += 1./np.log2(i+2)
    ndcg = dcg/idcg
    return ndcg
```

```
def mrr_for_user(self, user, k , modified_metric = False):
    true = self.test_item[user]
    pred = self.predict_item[user]
    relevant_items = [1 if elem in true else 0 for elem in pred]
    if sum(relevant_items) == 0:
        return 0.
    else:
        index_of_first_relevant = relevant_items.index(1)
        return 1. / (index_of_first_relevant +1)

def calculate_metric_for_rating(self, metric_function, metric_per_user=False):
          if self.calculate_rating_metrics == False:
        print('calculate_rating_metrics == False')
        if metric_per_user:
            return -1 , {}
        else:
            return -1

    user_metric = {}
    sum_metric_for_all_users = 0.

    for user in self.users:
        real_ratings = self.common_rating_test[user]
        predicted_ratings = self.common_rating_predict[user]

        metric = 0.
        if len(real_ratings) > 1:
            metric = metric_function(real_ratings, predicted_ratings)
        user_metric[user] = metric
        sum_metric_for_all_users += metric

    result_metric = sum_metric_for_all_users/len(self.users)

    if metric_per_user:
        return result_metric, user_metric
    else:
        return result_metric
```

```
    def RMSE(self, metric_per_user=False):
        return self.calculate_metric_for_rating(self.rmse_for_user, metric_per_user)
    def MAE(self, metric_per_user=False):
         return self.calculate_metric_for_rating(self.mae_for_user, metric_per_user)

    def MSE(self, metric_per_user=False):
        return self.calculate_metric_for_rating(self.mse_for_user, metric_per_user)

    def rmse_for_user(self, real_ratings, predicted_ratings):
        return np.sqrt(np.mean((real_ratings - predicted_ratings)**2))

    def mse_for_user(self, real_ratings, predicted_ratings):
        return np.mean((real_ratings - predicted_ratings)**2)

    def mae_for_user(self, real_ratings, predicted_ratings):
        return np.mean(np.absolute(real_ratings - predicted_ratings))

class Implicit(object):

    def __init__(self):

        self.model = None
        self.mapped_trainset = None
        self.mapping_dict = None
        self.inv_mapping_dict = None
        self.max_index_of_item = None
        self.max_index_of_user = None
        self.item_users = None
        self.user_items = None
        self.k = 10
        self.param = None
            self.default_param = {'factors': 100, 'regularization':0.01, 'iterations':15,'use_native':True, 'use_cg':True,
'use_gpu':False,'calculate_training_loss':False, 'num_threads':0}

        self.mean_user_factors = None
        self.mean_item_factors = None
        self.baseline_recommend_items = None
        self.baseline_recommend_scores = None
```

```
    def fit_trainset(self, raw_train_dataset):
        trainset = copy.deepcopy(raw_train_dataset)
        #trainset = trainset.drop_duplicates(subset=['user','item'])
        self.mapping_dict, self.inv_mapping_dict = fit_coder(trainset, 'user', 'item', 'rating')
        self.mapped_trainset = code(copy.deepcopy(trainset), 'user','item','rating',self.mapping_dict)
        self.max_index_of_item = len(self.mapped_trainset.item.unique())
        self.max_index_of_user = len(self.mapped_trainset.user.unique())
        row = self.mapped_trainset.item.values
        col = self.mapped_trainset.user.values
        data = self.mapped_trainset.rating.values
        self.item_users = csr_matrix( (data,(row,col)), shape=(self.max_index_of_item,self.max_index_of_user))
        self.user_items = self.item_users.T.tocsr()
        self.user_items = bm25_weight(self.user_items, B=0.7).tocsr()*5
        self.item_users = self.user_items.T.tocsr()
        # #Experiment ----------
        # add_one = self.item_users.toarray() + 1
        # self.item_users = csr_matrix(add_one)
        # # ------------------------
        self.user_items = self.item_users.T.tocsr()

    def add_fit_trainset(self, new_raw_train_dataset):
        if self.mapped_trainset is None:
            self.fit_trainset(new_raw_train_dataset)
        else:
            new_trainset = copy.deepcopy(new_raw_train_dataset)
            new_train = code(copy.deepcopy(new_trainset), 'user','item','rating',self.mapping_dict)
            ind_item = new_train[new_train.item.isnull()].index
            ind_user = new_train[new_train.user.isnull()].index
            unknown_items = new_trainset.loc[ind_item,'item'].unique()
            unknown_users = new_trainset.loc[ind_user,'user'].unique()
            len_new_items = len(unknown_items)
            len_new_users = len(unknown_users)

                new_item_dic = {key: value for key, value in zip(unknown_items,range(self.max_index_of_item,
self.max_index_of_item+len_new_items))}
                 new_user_dic = {key: value for key, value in zip(unknown_users,range(self.max_index_of_user,
self.max_index_of_user+len_new_users))}
```

```
            inv_new_item_dic = {value: key for key, value in zip(unknown_items,range(self.max_index_of_item, self.max_index_of_item+len_new_items))}
            inv_new_user_dic = {value: key for key, value in zip(unknown_users,range(self.max_index_of_user, self.max_index_of_user+len_new_users))}

        self.max_index_of_item += len_new_items
        self.max_index_of_user += len_new_users
        self.mapping_dict['item'].update(new_item_dic)
        self.mapping_dict['user'].update(new_user_dic)
        self.inv_mapping_dict['item'].update(inv_new_item_dic)
        self.inv_mapping_dict['user'].update(inv_new_user_dic)
        #self.mapped_trainset = self.mapped_trainset.append(new_trainset, ignore_index=True)

        self.mapped_trainset = pd.concat([self.mapped_trainset, new_mapped_trainset], ignore_index=True)
        self.mapped_trainset = self.mapped_trainset.drop_duplicates(subset=['user','item'])

        row = self.mapped_trainset.item.values
        col = self.mapped_trainset.user.values
        data = self.mapped_trainset.rating.values

        self.item_users = csr_matrix( (data,(row,col)), shape=(self.max_index_of_item,self.max_index_of_user))
        self.user_items = self.item_users.T.tocsr()

            factors_for_new_unknown_users = [list(self.mean_user_factors)]*len_new_users
            if len(factors_for_new_unknown_users) > 0:
                                                                    self.model.user_factors =
np.concatenate([self.model.user_factors,factors_for_new_unknown_users])

            factors_for_new_unknown_items = [list(self.mean_item_factors)]*len_new_items
            if len(factors_for_new_unknown_items) > 0:
                                                                    self.model.item_factors =
np.concatenate([self.model.item_factors,factors_for_new_unknown_items])

            print('factors extended')
    def set_k(self, k):
        self.k = int(k)
```

```
    def fit_model(self, dic_param = {}, fit_new_model = True):
        if self.item_users is None:
            print('Firstly fit trainset')
        else:
            if fit_new_model == False:  #check if available previous model
                if not self.model:
                    fit_new_model = True
            if fit_new_model:
                d = copy.deepcopy(self.default_param)
                d.update(dic_param)
                self.param = d
            else:
                d = copy.deepcopy(self.param)
                d.update(dic_param)
                if d['factors'] != self.param['factors']:
                    print('different amount of facors! Previous: '+str(self.param['factors'])+'; Now: '+str(d['factors']) +';
Fit new model')
                    fit_new_model = True
            if fit_new_model:
                                                                                    self.model        =
AlternatingLeastSquares(factors=d['factors'],regularization=d['regularization'],iterations=d['iterations'],
use_native=d['use_native'],use_cg=d['use_cg'],use_gpu=d['use_gpu'],calculate_training_loss
=d['calculate_training_loss'], num_threads=d['num_threads']) #dic_param
            else:
                previous_user_factors = self.model.user_factors
                previous_item_factors = self.model.item_factors
                                                                                    self.model        =
AlternatingLeastSquares(factors=d['factors'],regularization=d['regularization'],iterations=d['iterations'],
use_native=d['use_native'],use_cg=d['use_cg'],use_gpu=d['use_gpu'],calculate_training_loss
=d['calculate_training_loss'], num_threads=d['num_threads']) #dic_param
                self.model.user_factors = previous_user_factors
                self.model.item_factors = previous_item_factors

            self.model.fit(self.item_users)
            self.mean_user_factors = self.model.user_factors.mean(axis=0)
            self.mean_item_factors = self.model.item_factors.mean(axis=0)

            scores = np.dot(self.model.item_factors,self.mean_user_factors)
```

```
            items = list(range(self.max_index_of_item))
            result = list(zip(scores, items))
            result.sort(reverse=True)
            recommend = np.array(result)

            self.baseline_recommend_items = [self.inv_mapping_dict['item'][int(item)] for _ , item in recommend]
            self.baseline_recommend_scores = [score for score , _ in recommend]

    def get_user_factors(self):
        real_user_factors = {}
        for i in range(self.max_index_of_user):
            real_user_factors[self.inv_mapping_dict['user'][i]] = list(self.model.user_factors[i])
        return real_user_factors

    def get_item_factors(self):
        real_item_factors = {}
        for i in range(self.max_index_of_item):
            real_item_factors[self.inv_mapping_dict['item'][i]] = list(self.model.item_factors[i])
        return real_item_factors
        def recommend_for_user(self, user_true_name, filter_already_liked_items = True, return_scores = False,
recalculate_user=False):
        if self.mapping_dict is None:
            print('Firstly fit_trainset')
            return None
        if self.model is None:
            print('Firstly fit_model')
            return None

        if user_true_name in self.mapping_dict['user'].keys():
            user = self.mapping_dict['user'][user_true_name]
                                                rec = self.model.recommend(user, self.user_items, self.k,
filter_already_liked_items=filter_already_liked_items, recalculate_user=recalculate_user)
            items = [self.inv_mapping_dict['item'][item] for item, _ in rec]
            scores = [score for _, score in rec]
            if return_scores:
                return items, scores
            else:
                return items
```

```
    else:
        items = self.baseline_recommend_items[:self.k]
        if return_scores:
            scores = self.baseline_recommend_scores[:self.k]
            return items, scores
        else:
            return items

        def recommend(self, users_list, filter_already_liked_items = True, return_scores = False, recalculate_user=False):
    if self.mapping_dict is None:
        print('Firstly fit_trainset')
        return None
    if self.model is None:
        print('Firstly fit_model')
        return None

    result_user_items = {}
    result_user_scores = {}

    for user_true_name in tqdm(users_list):
        if return_scores:
            items, scores = self.recommend_for_user(user_true_name, filter_already_liked_items, return_scores, recalculate_user)
            result_user_items[user_true_name] = items
            result_user_scores[user_true_name] = scores
        else:
                items = self.recommend_for_user(user_true_name, filter_already_liked_items, return_scores, recalculate_user)
            result_user_items[user_true_name] = items
    if return_scores:
        return result_user_items, result_user_scores
    else:
        return result_user_items
```

ДОДАТОК Б Презентаційні матеріали

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рекомендаційна система товарів
Goods recommendation system

Науковий керівник:
к.т.н., доцент
Дідковська М.В.

Виконав:
студент групи КА-51
Тарнавський Максим

Основні визначення

Рекомендаційні системи - це набір технік інформаційного фільтрування, які пропонують користувачам потенційно корисні для них товари.

Актуальність

Торгівельний гігант **Amazon** заявив, що **35% свого доходу** він отримав завдяки їхнім рекомендаційним системам.

Більше **75% контенту**, який дивились користувачі компанії **Netflix**, був саме запропонований їхньою рекомендаційною системою.

Мета, предмет та об'єкт

Метою роботи є розробка рекомендаційної системи для вибору фільмів.

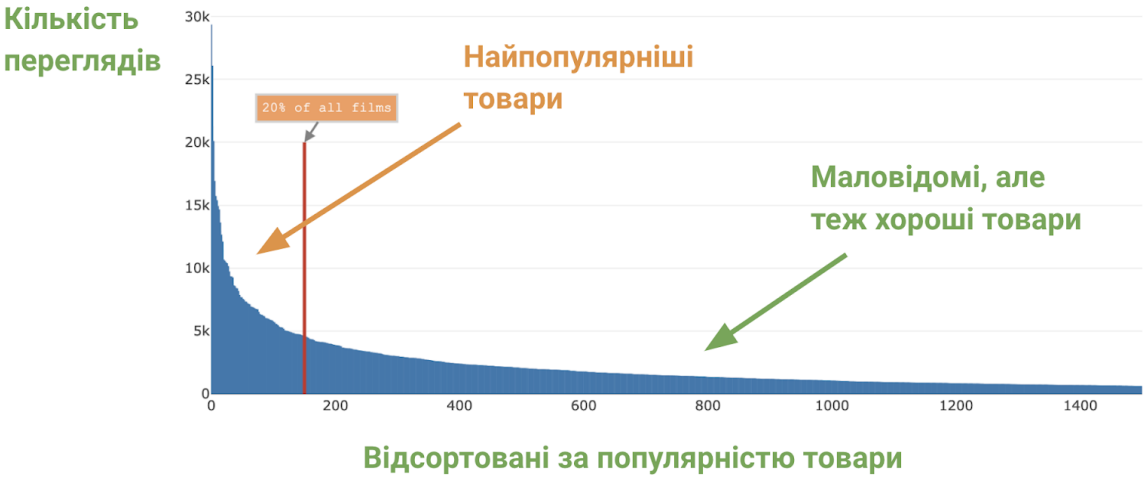
Об'єктом дослідження є рекомендаційні системи товарів.

Предметом дослідження є методи та алгоритми формування рекомендацій.

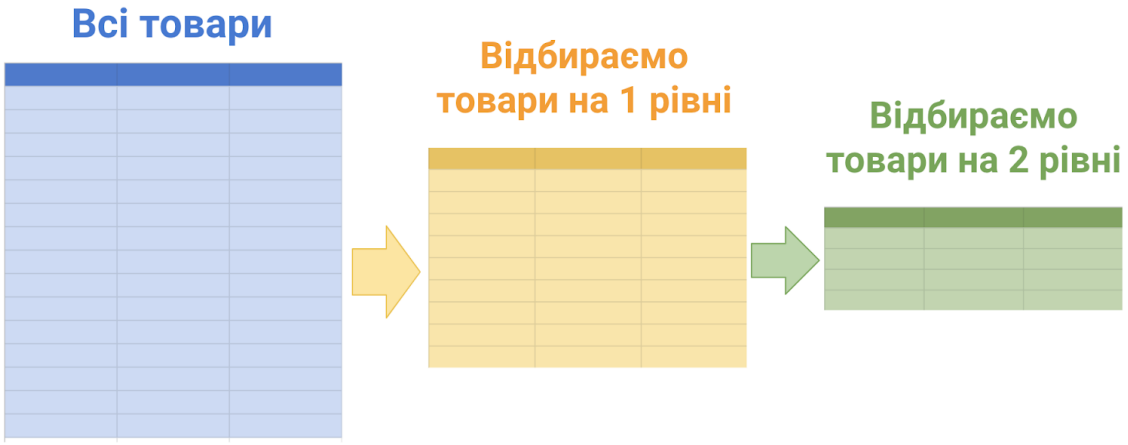
Постановка задачі

- провести дослідження існуючих методів формування рекомендацій
- розробити власний алгоритму формування рекомендацій на основі дворівневої моделі
- провести оцінку якості роботи запропонованого алгоритму
- програмно реалізувати рекомендаційну системи на основі розробленого алгоритму

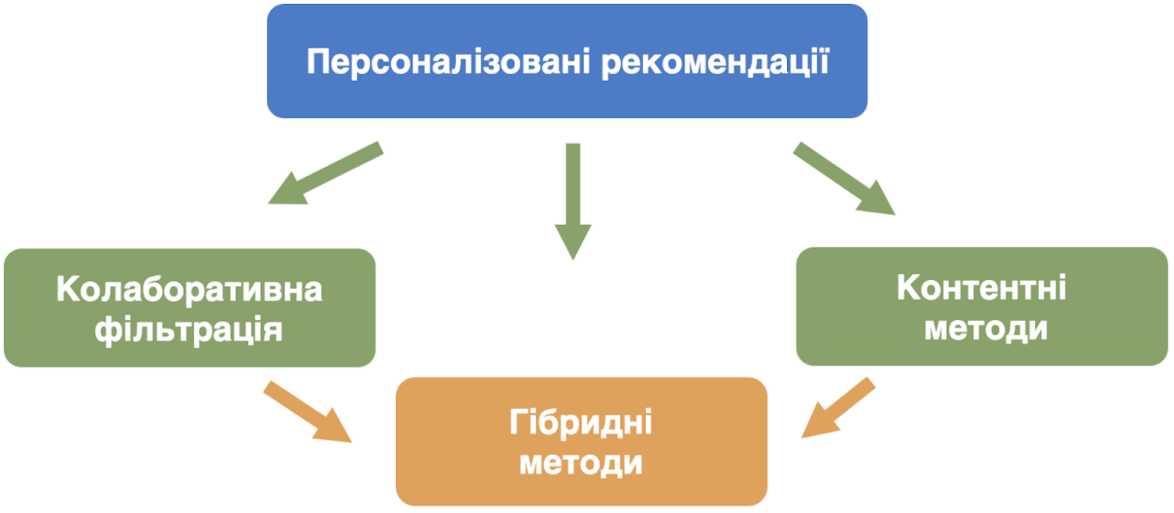
Існуючі підходи до розв’язку та їхні недоліки



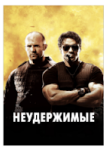
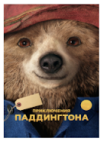
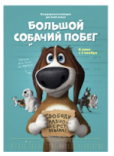
Алгоритм



Існуючі підходи до розв’язку та їхні недоліки

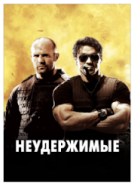


Матриця взаємодій між користувачами та товарами

					
	2	5	3		?
	5	2		4	?
	5	2	4		?

Функціональна залежність вподобання користувача

$$f_u : I \to \mathbb{R}$$

$$f_u\left(\ \text{  }\ \right) = 4.5$$

$$f_u\left(\ \text{  }\ \right) = 2.7$$

Існуючі підходи до розв’язку та їхні недоліки

Колаборативна фільтрація

+ Швидкість
+ Нескладні в реалізації
+ Потребують мінімальний об’єм інформації

- Не враховують додаткову інформацію
- Проблема холодного старту

Контентні методи

+ Більша релевантність рекомендацій
+ Можливість навчатись на підвибірці користувачів

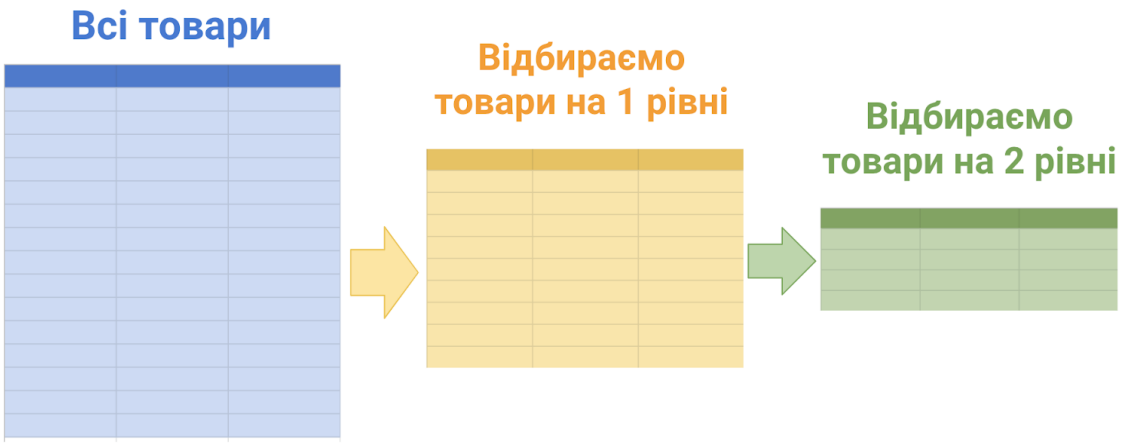
- Значно повільніші
- Ресурсоємні
- Потребують додаткову інформацію

Існуючі підходи до розв’язку та їхні недоліки

Гібридні методи

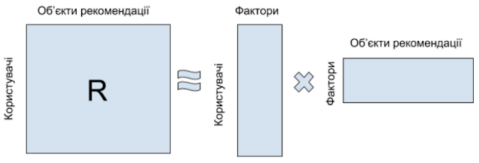
+ Поєднують переваги обох підходів, пропонуючи хороші рекомендації за прийнятний час

Запропонований алгоритм



Модель 1 рівня

Матрична факторизація



$u \in U \quad i \in I$ - множини користувачів та товарів

$x_u \in R^f \quad y_i \in R^f$ - фактор вектори, які описують користувача **u** та товар **i**

r_{ui} - міра вподобання користувачем **u** товару **i**

p_{ui} - індикатор наявності взаємодії між користувачем **u** товару **i**

$c_{ui} = 1 + \alpha r_{ui}$ Мета: $\hat{p}_{ui} = x_u^T y_i$

$$min_{x_*, y_*} \Sigma_{u,i} c_{ui} (p_{ui} - x_u^T y_i)^2 + \lambda (\Sigma_u \|x_u\|^2 + \Sigma_i \|y_i\|^2)$$

Модель 2 рівня

Градієнтний бустинг дерев рішень

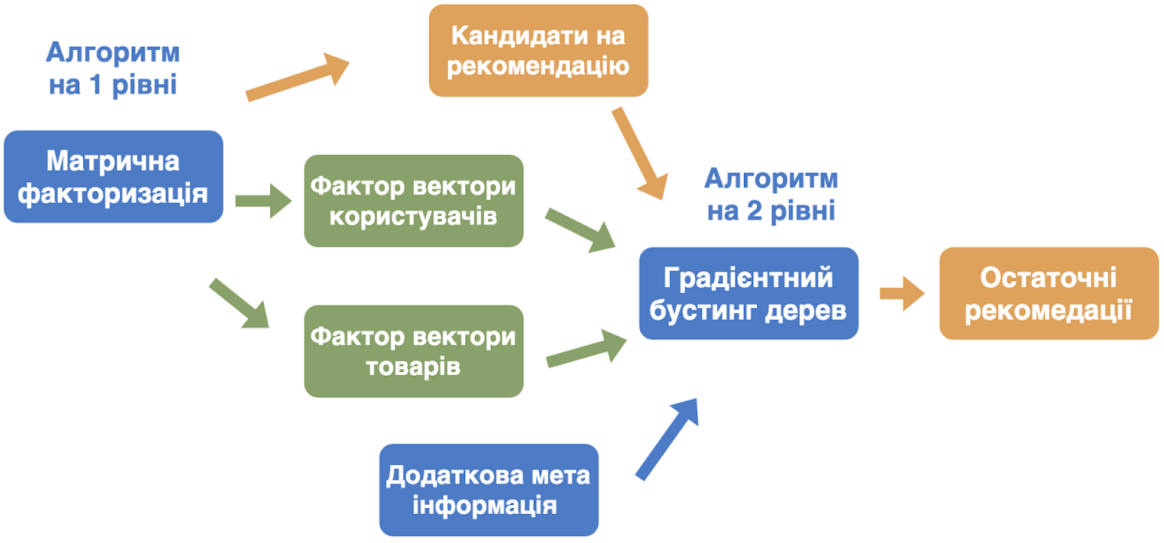
Дерева рішень зазвичай представляють собою бінарні дерева, в яких в кожній внутрішній вершині записана умова, а в кожному листі дерева - прогнозоване значення

$X_\ell = \{x \in X_m | [x^j \le t]\}$ $X_r = \{x \in X_m | [x^j > t]\}$ - поділ умовою на підмножини

$Q(X_m, j, t) = \frac{|X_\ell|}{|X_m|} H(X_\ell) + \frac{|X_r|}{|X_m|} H(X_r)$ - умову підбирають так, щоб мінімізувати критерій похибки

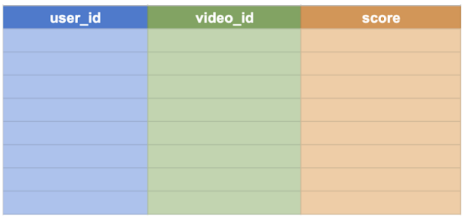
$H(X) = \sum_{k=1}^{K} p_k(1 - p_k)$ - критерій інформативності Джині $p_k = \frac{1}{X} \sum_{i \in X} [y_i = k]$

Запропонований алгоритм

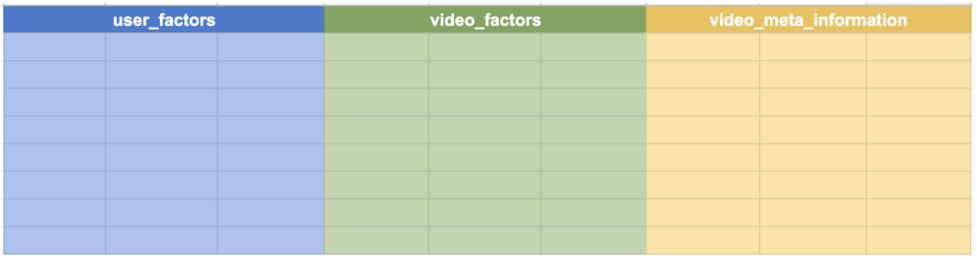


Перехід від моделі 1 рівня до моделі 2 рівня

Набір кандидатів від моделі 1 рівня:



Вхідні дані для моделі 2 рівня:



Критерії порівняння

Online метрики

Бізнес показники, такі як:
- Кількість покупок
- Час проведений в системі
- CTR - відношення числа кліків до числа показів

Offline метрики

Для прогнозу оцінки:
- RMSE
- MAE

Для оцінки якості рекомендації:
- Precision
- MAP
- NDCG

Критерії порівняння

$$RMSE = \frac{1}{N}\sum_{ui}\sqrt{(r_{ui} - \hat{r}_{ui})^2} \qquad MAE = \frac{1}{N}\sum_{ui}|r_{ui} - \hat{r}_{ui}|$$

r_{ui} -істинне значення оцінки

$\hat{r}_{ui}$ -прогнозоване значення оцінки

$Relevance(k)$ - індикатор релевантності рекомендованого товару на k-тій позиції, може набувати значення {0,1}

$$Precision@K = \frac{1}{K}\sum_{k=1}^{K} Relevance(k)$$

$$Average\ Precision@K = \frac{1}{K}\sum_{k=1}^{K} Relevance(k) \cdot (Precision@k)$$

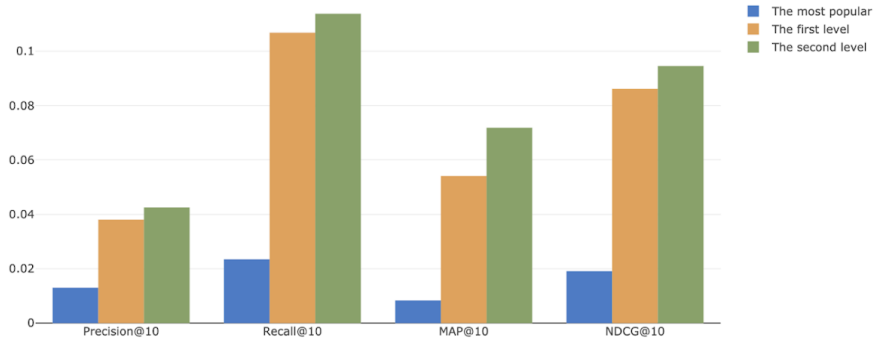
$$MAP@K = \frac{1}{N}\sum_{j=1}^{N} Average\ Precision@K_j$$

$$DCG@K = \sum_{k=1}^{K} \frac{Relevance(k)}{log_2(k+1)}$$

$$NDCG@K = \frac{DCG@K}{max(DCG@K)}$$

Порівняльний аналіз практичних результатів

	Precision@10	Recall@10	MAP@10	NDCG@10
The second level	0.0425413	0.113818	0.0718531	0.0945739
The first level	0.038066	0.106846	0.0541142	0.0861713
The most popular	0.0130161	0.0234716	0.00832245	0.0191037



Порівняльний аналіз практичних результатів

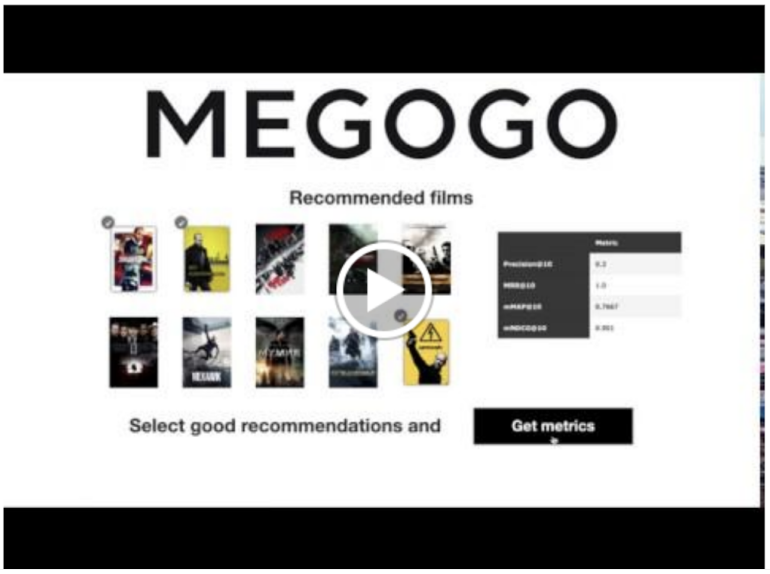
Результат дворівневої моделі 0.07185

Результат 1 місця в змаганні 0.07107

#	△pub	Team Name	Kernel	Team Members	Score	Entries	Last
1	—	x0x0w1			0.07107	35	2mo
2	—	AfterParty			0.06882	51	2mo
3	—	Netflix			0.06301	39	2mo
4	—	Closer			0.06139	50	2mo
5	—	return_0			0.06035	34	2mo
6	—	zteam			0.05963	25	2mo

Моя команда на момент змагання

Демонстрація роботи продукту



Висновки

- запропоновано алгоритм дворівневої моделі для здійснення рекомендацій, який поєднує собі матричну факторизацію та градієнтний бустинг дерев
- розроблений алгоритм отримав значення метрики MAP@10 **0.07185** в змаганні Megogo, що перевищує найкращий результат запропонований на момент змагання
- даний алгоритм впроваджено в компанії **Ciklum**

Шляхи подальшого розвитку

Варто розглянути :

- DSSM нейронні мережі
- Тензорна факторизація матриці

Дякую за увагу !